Microsoft® BASIC

Professional Development System

Getting Started

Microsoft®

Guide to Keystrokes

The following table lists the most commonly used keystrokes in the QuickBASIC Extended (QBX), Programmer's WorkBench (PWB), and CodeView environments. Unless stated otherwise, each keystroke has the same function in all three environments.

Alt	Activate the menu bar. Press the highlighted characters in the menu names to open menus and to choose commands from a menu. Or, use the Left Arrow and Right Arrow keys to highlight a menu title, and then press the Up Arrow and Down Arrow keys to choose a command. Pressing Alt a second time closes all open menus and deactivates the menu bar.
Esc	Exit Help. Esc also closes menus and dialog boxes, cancelling any entries.
F1	Invoke the online Help system. To get help, move the cursor to the topic on which you want more information: a keyword, a command, a dialog box, or an error message. Then press F1. To exit Help press Esc.
Shift + F1	In the QBX environment, Shift + F1 explains Help. In PWB and CodeView, Shift + F1 displays the Help Contents.
F4	Display the output screen from within the environment. Pressing F4 a second time returns you to the environment.
F5	Begin executing the currently loaded program.
F6	Move the cursor between visible windows.
Shift + Arrow	Select text.
Ctrl + Ins	Copy selected text.
Shift + Del	Cut selected text.
Shift + Ins	Paste cut or copied text at the current cursor position.
Alt + Backspace	Undo the previous edit.
Ctrl + Backspace	Restore the edit you've undone.

Microsoft® BASIC

Getting Started

Version 7.1
For IBM® Personal Computers
and Compatibles

Microsoft Corporation

Simultaneously published in the U.S. and Canada.

Printed and bound in the United States of America.

Document No. DB06823-0590
OEM No. D704-7Z

Table of Contents

Figures

Introduction

Welcome to Microsoft® BASIC version 7.1 Professional Development System. Microsoft BASIC includes the tools and files you'll need to create and debug large, sophisticated BASIC programs that run under the DOS and OS/2 operating systems. The Microsoft BASIC package includes the following:

- Microsoft QuickBASIC Extended (QBX) development environment
- Microsoft Programmer's WorkBench (PWB) development environment
- BASIC compiler (BC)
- Integrated Indexed Sequential Access Method (ISAM) package
- Major utilities:
  - Microsoft Segmented-Executable Linker (LINK)
  - Microsoft Library Manager (LIB)
  - Microsoft Program-Maintenance Utility (NMAKE)
  - Run-Time Module Build Utility (BUILDRTM)
  - Microsoft Help File Creation Utility (HELPMAKE)
  - Microsoft QuickHelp
- Add-on libraries: Date/Time, Financial, and Format
- Toolboxes: Matrix Math, Presentation Graphics, and User Interface
- Microsoft CodeView® debugger and other mixed-language development tools
- Printed and online documentation

Distribution of Run-Time Modules

As described in Paragraph 5 of the Microsoft License Agreement included with this product, the following run-time modules may be distributed under the terms and conditions listed in the agreement:

BRT71xxx.EXE, BRT71xxx.DLL, HELVB.FON, TMSRB.FON, MSHERC.COM, MOUSE.COM, PATCH87.EXE, FIXSHIFT.COM, PROISAM.EXE, PROISAMD.EXE, ISAMIO.EXE, ISAMCVT.EXE, ISAMPACK.EXE, and ISAMREPR.EXE.

These modules can only be distributed in conjunction with and as a necessary and integrated component of your software product, in such a manner that they enhance or supplement the core value otherwise existing in your software product.

Documentation

Microsoft BASIC 7.1 includes a full set of printed and online documentation.

Printed Documentation

Getting Started gets you up and running with Microsoft BASIC.

The *Programmer's Guide* provides information about programming concepts and techniques. The first part of the manual focuses on the features of the BASIC language, including such topics as control-flow structures, string processing, and error/event handling. The second part of the manual describes program development utilities supplied with Microsoft BASIC. These chapters give the syntax and command-line options for each utility, as well as illustrative examples. The appendixes contain the BASIC language definitions.

The *BASIC Language Reference* is divided into three parts. Each part begins with summary tables listing the functions and statements described in that part, followed by individual reference descriptions in alphabetical order.

- Part 1, "Language Reference," describes Microsoft BASIC functions, statements, and metacommands.
- Part 2, "Add-On-Library Reference," describes the functions in the add-on libraries.
- Part 3, "BASIC Toolbox Reference," describes the procedures in the BASIC toolboxes.

The appendixes describe keyboard/scan codes, reserved words, command-line tools, and error messages.

Online Documentation

Microsoft BASIC puts a complete online reference database at your fingertips with the new Microsoft Advisor online Help system. Online Help gives you context-sensitive Help for the BASIC language, QBX, command-line compiler options, error messages, and symbols in your programs. See Chapter 6, "Using Online Help," for more information.

Document Conventions

Microsoft documentation uses the term "OS/2" to refer to the OS/2 systems—Microsoft Operating System/2 (MS® OS/2) and IBM OS/2. Similarly, the term "DOS" refers to the MS-DOS® and IBM Personal Computer DOS operating systems. The name of a specific operating system is used when it is necessary to note features that are unique to that system.

This manual uses the following typographic conventions:

Example of convention	Description
`setup`	Words you type appear in this boldface font.
BASIC.LIB, ADD.EXE, COPY, LINK	Uppercase (capital) letters indicate filenames and operating-system environment variables and commands.
SUB, **IF**, **LOOP**, **PRINT**, **TIME$**	Bold uppercase letters indicate language-specific keywords with special meaning to Microsoft BASIC or another Microsoft language.
`Bad File Mode`	This font is used for error messages.
handles	Italic letters indicate placeholders for information you supply. Italics are also occasionally used in the text for emphasis.
[[*disksize*]]	Items inside double square brackets are optional.
{**WHILE**|**UNTIL**}	Braces and a vertical bar indicate a mandatory choice between two or more items. You must choose one of the items unless all of the items also are enclosed in double square brackets.
Ctrl key	Initial capital letters are used for the names of keys and key sequences, such as Enter and Ctrl+R. The key names used in this manual correspond to the names on the IBM Personal Computer keyboard. Other machines may use different names.
Alt+F1	A plus (+) indicates a combination of keys. For example, Alt+F1 means to hold down the Alt key while pressing the F1 key.
Down Arrow key	The cursor movement ("arrow") keys are called direction keys. Individual direction keys are referred to by the direction of the arrow on the key top (Left, Right, Up, or Down).

"defined term"	Quotation marks usually indicate a new term defined in the text.
Video Graphics Array (VGA)	Acronyms are usually spelled out the first time they are used.

Chapter 1
New Features

Microsoft BASIC combines powerful new language features with capacity, performance, and usability enhancements to provide a high-productivity development system for professional BASIC programmers. This chapter describes the new features for versions 7.0 and 7.1. Changes for version 7.1 are indicated by the words "Version 7.1" in the left margin.

New Features in Version 7.1

Microsoft BASIC version 7.1 extends BASIC's power under OS/2, improves its compatibility with other languages, and provides new language features:

- BASIC's Indexed Sequential Access Method (ISAM) is now fully supported for OS/2.
- The Programmer's WorkBench (PWB) provides a sophisticated development environment for mixed-language programming under DOS or OS/2.
- The CodeView debugger version 3.0 contains many improvements to simplify the task of debugging large BASIC and mixed-language programs under DOS or OS/2.
- QuickHelp provides access to the Microsoft Advisor Help system from the command line.
- The new **REDIM PRESERVE** statement lets you change the size of an array without erasing the data it contains.
- BASIC now supports passing parameters by value within BASIC procedures and passing arrays containing fixed length strings as parameters.
- BASIC is fully compatible with Microsoft C version 6.0.

The rest of this chapter discusses the features new to Microsoft BASIC versions 7.0 and 7.1 in greater detail.

Language Enhancements

Microsoft BASIC contains major language enhancements designed for business, scientific, and general-purpose BASIC programming.

Integrated ISAM

The Indexed Sequential Access Method package provides a fast and simple method for accessing specific records in large and complex data files. Microsoft BASIC integrates all ISAM statements and functions with the BASIC language. BASIC ISAM includes statements for transaction processing and for data retrieval and manipulation.

Version 7.1 In version 7.1, ISAM statements and functions are supported for OS/2 as well as DOS.

Microsoft BASIC also includes several utilities for use with ISAM files created with BASIC:

- ISAMCVT.EXE (DOS only) converts Btrieve and dBASE files for use with ISAM.
- ISAMREPR.EXE repairs corrupted ISAM databases.
- ISAMPACK.EXE compacts ISAM files to save disk space.
- ISAMIO.EXE converts ASCII files to ISAM format and vice versa.

For more information about ISAM, see Chapter 10, "Database Programming with ISAM," in the *Programmer's Guide*.

DOS File Management

The **DIR$** and **CURDIR$** functions and the **CHDRIVE** statement make it easier to manage DOS files from BASIC programs.

DIR$** is similar to the DOS DIR command, except that the filenames are returned one at a time. **CURDIR$ returns the current directory specification, and **CHDRIVE** changes the current drive.

Currency Data Type

The currency data type maintains to-the-penny precision while providing the speed of integer math for programming accounting tasks. Its internal representation as an integer gives this type a significant advantage in speed over the floating-point data type for operations such as addition and subtraction. For more information about the currency data type, see Chapter 15, "Optimizing Program Size and Speed," in the *Programmer's Guide*.

Procedure-Level Error Handling

Microsoft BASIC contains local error trapping and handling for procedures, making error handling much more efficient. In previous versions of BASIC, error-handling routines existed at the module level. When a handler was turned on, it was active for all procedures within the module.

With Microsoft BASIC, you can create both module-level and procedure-level error handlers. The same error can invoke different error-handling routines, depending on which procedure is running. For example, you may want to invoke different error-handling routines for **ERR** code 54, `Bad File Mode`, because the error has different meanings for different file operations. For more information on error handling, see Chapter 8, "Error Handling," in the *Programmer's Guide*.

Static Arrays in Records

In previous versions of BASIC, you could create user-defined data structures that contained numeric and fixed-string data types by using the **TYPE...END TYPE** statements. Now you can use static arrays in addition to numeric and string data types, which gives you more flexibility in building data structures.

Preserve Data when Redimensioning an Array

Version 7.1 By adding the **PRESERVE** option to the **REDIM** statement, you can preserve the data that exists in an array when changing its outer boundary. This simplifies the dynamic control of the amount of memory consumed by an array. For details and an example, see the online Help on the **REDIM** statement .

Improved Parameter Passing

Version 7.1 With previous versions of BASIC, you could emulate passing parameters by value by surrounding the parameter with parentheses. Now you can use the **BYVAL** keyword in **DECLARE**, **SUB**, and **FUNCTION** statements for BASIC procedures. For details on passing parameters by value, see the online Help for the **DECLARE**, **SUB**, and **FUNCTION** statements.

BASIC now also supports passing arrays containing fixed-length strings as parameters. For more information, see the online Help for the **SUB** and **FUNCTION** statements.

Improved COM Support with ERDEV$ and ERDEV

With previous versions of BASIC, if you had a device timeout error, there was no way to find out which device timed out, or which control line caused the timeout. With Microsoft BASIC, information about the timeout is available through the **ERDEV$** and **ERDEV** functions. **ERDEV$** indicates whether a timeout is occurring on the communications port. **ERDEV** indicates the type of timeout error that occurred (that is, CTS, DSR, or DCD control-line errors). See the **OPEN COM** statement in the *BASIC Language Reference* for more information.

Date/Time, Financial, and Format Add-On Libraries

With these new libraries, you can use BASIC to perform spreadsheet-style calculations and formatting. For example, you can compute the number of days between two dates by converting the dates from month-day-year format to number format, and then subtracting. You can use financial functions to calculate double-declining balance depreciation, future value, net present value, internal rate of return, and other common financial calculations. For more information about Date/Time, Financial, and Format add-on libraries, see Part 2 of the *BASIC Language Reference*.

Matrix Math Toolbox

The new Matrix Math toolbox contains sample BASIC code for several common matrix math operations, including addition, subtraction, multiplication and division, matrix inversion, determinant calculation, and solution of simultaneous equations using Gaussian elimination. For more information about the Matrix Math toolbox, see Part 3 of the *BASIC Language Reference*.

Presentation Graphics Toolbox

You can use the Presentation Graphics toolbox to display charts and graphs from your programs. The Presentation Graphics toolbox includes procedures for pie charts, bar and column charts, line graphs, and scatter diagrams. Figure 1.1 shows an example of a column chart created with this toolbox.

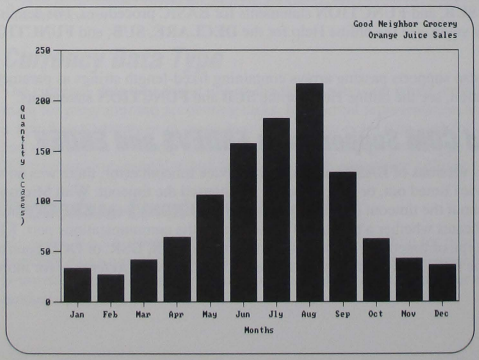

Figure 1.1 Presentation Graphics

The toolbox also includes a set of special graphics fonts. For more information about Presentation Graphics, see Chapter 6, "Presentation Graphics," in the *Programmer's Guide*.

User Interface Toolbox

With the new user interface code samples, you can design your own user interface using BASIC procedures. The code samples in this toolbox give you complete control of character-based window interfaces. For example, you could write a BASIC program with multiple windows, menu bars, dialog boxes, and mouse interaction. For more information about the User Interface toolbox, see Part 3 of the *BASIC Language Reference*.

C Version 6.0 Compatibility

Version 7.1 BASIC incorporates the C version 6.0 startup code. This makes BASIC fully compatible with the latest version of C. However, this can cause problems for mixed-language programmers who have earlier versions of C, FORTRAN, or Pascal. Compatible run-times for these languages are available free from Microsoft. Contact Product Support Services for more information.

Capacity and Performance Enhancements

Microsoft BASIC eliminates many of the capacity and performance barriers that formerly limited the size and sophistication of BASIC applications.

Far String Support

In previous versions of BASIC, all variable-length string data was stored in near memory. This relatively small portion of memory (64K maximum) also stores the rest of the simple variables—integers, floating-point numbers and fixed strings, all constants, the stack, and some run-time overhead. Even when the only variables you use are variable-length strings, your maximum data storage is limited.

Microsoft BASIC supports "far strings," which enable you to store variable-length string data outside of near memory in multiple segments of far memory. This gives you a full 64K of string storage in the module level of the main module, plus several additional 64K blocks of storage, depending on how you write your program. Removing variable-length strings from near memory creates significantly more room for other simple variables. For more information on using far strings, see Chapter 11, "Advanced String Storage," and Chapter 13, "Mixed-Language Programming with Far Strings," in the *Programmer's Guide*.

Overlay Support

With overlays, you can write and run programs up to almost 16 megabytes in size when compiling and linking from the command line. In an overlaid version of a program, specified parts of the program are only loaded if and when they are needed. Specifying overlays can be useful if you have compiled a program that is too large to load in memory. For more information about overlays, see Chapter 18, "Using LINK and LIB," in the *Programmer's Guide*.

Improved Granularity

The extent to which BASIC run-time routines are divided into individually accessible pieces is called "granularity." With Microsoft BASIC's improved granularity, you can link with the minimum amount of library code needed to produce stand-alone programs. This gives you smaller compiled programs, saving disk space and memory. For more information, see Chapter 15, "Optimizing Program Size and Speed," in the *Programmer's Guide*.

Optimized Code Generation

Optimized code generation automatically makes compiled BASIC programs smaller and faster. Also, using a new compiler switch, you can tell the compiler to generate code for the 80286 (and later) microprocessor family, taking advantage of that family's machine instructions. For more information on optimized code generation, see Chapter 15, "Optimizing Program Size and Speed," in the *Programmer's Guide*.

Improved Alternate Math Package

For target systems without an 80x87 math coprocessor, the alternate math package gives programs greatly improved speed. For example, if your application involves spreadsheet-style math without recursive calculations, using the alternate math package may give you a noticeable performance increase. For more information about the alternate math package, see Chapter 15, "Optimizing Program Size and Speed," in the *Programmer's Guide*.

Improved IEEE Math Coprocessor Emulation

For target systems with or without a math coprocessor, an improved coprocessor emulator gives you faster high-precision calculations.

Improved Code Generation

Version 7.1 Relative jumps and short jumps to near targets are optimized, resulting in smaller, faster programs. This improvement affects all control statements.

Environment and Usability Enhancements

Microsoft BASIC offers two sophisticated development environments: Microsoft QuickBASIC Extended (QBX) and Microsoft Programmer's WorkBench (PWB). Both provide the following advanced features:

- Integrated debugging capabilities within the environment and compatibility with CodeView version 3.0 for debugging and optimizing executable programs at a variety of levels
- Support for expanded memory specification (EMS 4.0) to create large executable programs
- Undo/Redo capabilities
- Customizable menus and key assignments
- Control of build options and target environment from the development environment
- A comprehensive online Help system designed for professional programmers

In addition to QBX and PWB, BASIC now includes the QuickHelp environment for viewing Microsoft Advisor online Help files.

The QuickBASIC Extended Environment

Microsoft QuickBASIC Extended (QBX) is an advanced development environment for programmers who write most or all of their code in BASIC. It provides line-by-line syntax checking and on-demand program execution without recompiling. QBX also provides the following key features:

- QBX automatically uses expanded memory, if present, for any parts of your program source code that are less than 16K in size. The View SUBs command now enables you to determine whether a program unit (module-level code or procedure) will fit into expanded memory.
- Historical Undo/Redo commands. You can use Undo to step back through your last 20 edits.
- A customizable Utility menu. You can use the Utility menu to run DOS command-line programs or a custom editor, or for command-line compiling and overlay linking. You can even assign each menu item a shortcut key.
- Customizable key assignments. If you prefer to use a set of editing commands other than the QBX defaults, you can change your key file to one of those provided, or you can create your own custom assignments.
- Print to multiple printers or to a file.
- Debug watch window capacity doubled (from 8 to 16 expressions).
- Complete control of compiler switches for compiling from the environment.

For information on using QBX, see Chapter 3,"Using QuickBASIC Extended."

The Programmer's WorkBench Environment

Version 7.1 Microsoft Programmer's WorkBench (PWB) is an advanced development environment for programmers who often work in languages other than BASIC or who want to create, build, and debug their programs under OS/2. PWB also offers the following key features:

- PWB runs in real or protected modes, so you can write BASIC programs under DOS or OS/2.
- You can write programs in any Microsoft professional-level language and mix languages, such as BASIC and C, within the environment.
- The Source Browser allows you to search selected files, list references and definitions, build a call tree and outline, and view a list of relationships among program symbols.
- You can step through compiler errors and PWB will indicate where each error occurred in the source code.
- PWB and CodeView are integrated, so you can switch from one to the other without exiting PWB.
- Comprehensive Help is available on all command-line utilities and Microsoft languages installed on your system.

For information on using PWB, see Chapter 4, "Using the Programmer's WorkBench." See the PWB online Help for detailed information on performing tasks and customizing the environment.

The CodeView Debugger

Version 7.1 The CodeView debugger allows you to debug programs under DOS or OS/2. This is especially useful when you are using assembly language routines in a BASIC program. CodeView version 3.0 offers the following new features:

- Comprehensive online Help on how to use CodeView. Help is also available on any installed Microsoft language or utility from within CodeView.
- Automatic use of extended and expanded memory, if available.
- Full integration with PWB and compatibility with programs created in QBX.

For introductory information on how to debug programs using this environment, see Chapter 5, "Using The CodeView Debugger." For detailed information on how to perform tasks in CodeView and for information on how CodeView evaluates BASIC expressions, see the CodeView online Help.

QuickHelp

Version 7.1 Microsoft QuickHelp reads Microsoft Advisor online Help files under DOS or OS/2. This provides you with an easy way to get Help on the command-line utilities and development environments without loading QBX, PWB, or CodeView.

If you are using OS/2 or Microsoft Windows™, you can run QuickHelp in one window and do your programming in another. This can be more convenient than using Help in the programming environment, which obscures part of the program you are working on.

For more information on using QuickHelp, see Chapter 6, "Using Help."

Chapter 2
Setup

System Requirements

Microsoft BASIC requires the following minimum configuration:

- An IBM Personal Computer XT or compatible running DOS version 3.0 or later, or OS/2 version 1.1 or later.
- A hard disk drive.
- A 1.2M 5.25-inch floppy disk drive. (360K and 720K disk versions of this product are available from Microsoft. Contact Product Support Services for more information.)
- 640K of available memory.

Microsoft BASIC supports the Microsoft Mouse and any other pointing devices fully compatible with the Microsoft Mouse.

Setup Overview

Your Microsoft BASIC package includes the SETUP.EXE file on Disk 1 (Setup). You can use Setup in one of two ways:

- Install all the files you need to run BASIC, including the QBX environment, mixed-language development tools, utilities, and libraries.
- Selectively install files. Typically, these might include libraries or tools you did not choose to install the first time you used Setup.

Setup offers you a series of choices, letting you specify where to install the files and what features and utilities you want to install. To conserve disk space, you may choose not to use all of BASIC's features at first. You can later run Setup to selectively install features you decide to add.

Important You must run Setup to use Microsoft BASIC successfully. Setup decompresses files on the distribution disks (using UNPACK.EXE), builds libraries, and performs other operations to give you a usable BASIC environment. You cannot run BASIC by merely copying files from the distribution disks.

Running Setup

Setup contains information and Help screens to guide you through the setup process. The Setup Main menu points you to the various Setup screens where you can:

- Specify paths and directories for installed files.
- Specify libraries and run-time modules.
- Specify ISAM support.
- Specify files to install (for selective installation of specific files or parts of Microsoft BASIC).

To run Setup, place Disk 1 (Setup) in disk drive A:, type **a:setup** at the system prompt, and then follow the instructions on your screen. Before running Setup, however, you should keep the following points in mind.

Time Required to Install

Set aside an uninterrupted block of time. You can exit Setup if you need to, but it will not retain any of the options you selected. Installing Microsoft BASIC requires anywhere from 30 minutes to three hours, depending on the speed of your processor, available disk space, and the options you chose. For best performance, you should remove any terminate-and-stay-resident (TSR) programs from memory before running Setup.

Setup requires less time if you start it with the /BATCH option, as follows:

```
A:SETUP /BATCH
```

If you use /BATCH, Setup will operate the same way that it does without the option, with one exception. Instead of building BASIC run-times and libraries automatically, SETUP /BATCH creates a batch file named BUILD.BAT. You must run this batch file from the command line to complete the installation process.

Amount of Disk Space Required

The amount of disk space required varies depending on which features you choose to install. However, Setup determines the exact amount of space you need before proceeding with installation. It then checks available disk space. If you lack sufficient space, Setup prompts you to change the number of files to install. If you do not have sufficient disk space and you choose to continue, the installation may not be successful.

Custom or Default Configuration

The first time you install BASIC, you may not be sure which features you want to use. If this is the case, you may want to install with the default settings. You can always change these settings by running Setup again. To install the default configuration, choose Install with Current Options from the Setup Main menu, as shown in Figure 2.1, and then follow the instructions on your screen.

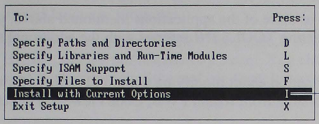

Figure 2.1 Install with Current Options

Setup Will Not Install Until You Tell It To

If you want to install only a few files or you want to change the configuration, you can browse through the various Setup screens and alter settings. You can spend as much time as you want; Setup takes no action until you choose Install with Current Options from the Setup Main menu.

Files Installed

During installation, Setup places a number of .EXE, .LIB and .OBJ files in the directories you specify, as well as documentation and sample-code files. Note that Setup may replace existing files. Therefore, it is a good idea to have Setup install files only to new directories, empty directories, or directories containing software you want to update.

Some of the .OBJ files are stub files. Depending on how you want to use BASIC, you might want to delete these files after installation. Before doing so, however, you should read the section "How Setup Uses Stub Files" later in this chapter.

After Running Setup

After you install Microsoft BASIC, you may want to do the following tasks:

- Create or modify the TOOLS.INI file
- Modify your environment variables
- Unpack individual files from the distribution disks

Creating or Modifying TOOLS.INI

Programmer's WorkBench, CodeView, QuickHelp, and NMAKE, all included with Microsoft BASIC, use the TOOLS.INI file for initialization information. Setup installs a file named TOOLS.PRE that contains preliminary settings for these applications. Setup places TOOLS.PRE in the \BC7\BINB directory by default.

If you don't have an existing TOOLS.INI file, you can rename TOOLS.PRE and place it in a directory that is indicated by your INIT environment variable.

If you already have a TOOLS.INI file and you have not previously installed any of these applications, you can append TOOLS.PRE to your existing TOOLS.INI file. For example:

```
C:\INIT>COPY TOOLS.INI + \BC7\BINB\TOOLS.PRE
```

If you have previously installed one or more of the applications mentioned above, you may want to selectively combine your existing TOOLS.INI settings with the additional settings in TOOLS.PRE.

The TOOLS.PRE file contains information about modifying the initial settings for each of the applications. Online Help is also available from within PWB, CodeView, and QuickHelp on the TOOLS.INI topic.

Modifying Your Environment Variables

Setup creates two files that you can use to change your environment variables: NEW-VARS.BAT for DOS, and NEW-VARS.CMD for OS/2. You can run these files from the command line, or you can append them to your AUTOEXEC.BAT file (DOS) or your CONFIG.SYS file (OS/2).

If you are installing for OS/2, you should also modify the LIBPATH line in your CONFIG.SYS file to include the directory where Setup installed BASIC's dynamic-link libraries (DLLs). By default, Setup places these files in the \BC7\BINP directory. Since LIBPATH is not part of the environment, NEW-VARS.CMD cannot modify it. If you change LIBPATH, you will have to restart your system in order for it to take effect.

Unpacking Individual Files

The files provided on the Microsoft BASIC distribution disks are compressed to conserve space. A dollar sign ($) is added to the file extension to show that these files must be unpacked before they can be used. The file PACKING.LST contains a table showing how the modified file extensions correspond to the unpacked files.

Setup automatically unpacks and installs all of the files you need to use the configuration you specified from the Setup menus. You can unpack individual files after Setup in order to modify your installation without rerunning Setup, to restore a file that was deleted, or to provide special support for mixed-language programming with Microsoft C version 6.0.

To unpack a file from a distribution disk, use the following syntax:

UNPACK *packed-filename unpacked-filename*

For example, to unpack the PWB.HLP file, insert the disk containing the file in drive A: and type:

```
UNPACK A:PWB.HL$ PWB.HLP
```

To find a particular file on the distribution disks, look at the file PACKING.LST on Disk 1.

Special Instructions for Microsoft C Users

If you have installed Microsoft C version 6.0, you should replace the PWB executable files and language extensions provided with C with the ones provided with this version of BASIC.

BASIC includes a later version of these files that supports the use of BASIC and C together in mixed-language programs.

The PWB extensions for the C language, PWBC.MXT and PWBC.PXT, are provided with BASIC, but are not installed by Setup. You should unpack these files individually from the BASIC distribution disks and use them to replace the C-language extensions provided with C version 6.0.

How Setup Uses Stub Files

This section explains how you can produce smaller stand-alone programs using stub files. If you are eager to use BASIC, you can skip this section for now and read it later.

A "stub file" replaces a BASIC library module with a module that performs little or no action. Stub files typically contain a small amount of code.

The advantage of stub files is that if you know that you will never use a certain feature in your program (such as support for VGA screen modes), you can use the stub file to eliminate code for that feature. The result is that your programs require less memory and disk space.

Stub files can be used for BASIC programs that use the BASIC run-time (BRT) module and for BASIC stand-alone executable files.

Stub Files and the BRT Module

BASIC programs can use the BRT module to support common routines. The use of the BRT module saves disk space by eliminating the need to add library routines to every executable file.

Setup uses stub files to build the BRT module. It uses a different stub file (see the following sections) for each feature you decide not to support.

Once installation is complete, most stub files have no further effect on the BRT module. You can delete them unless you plan to create stand-alone executable files, as described in the next section.

Stub Files and Stand-Alone Programs

You can produce stand-alone programs with the BASIC compiler by using the /O option. The advantage of stand-alone programs is that the BRT module need not be present for the program to run.

Do not delete stub files if you want to produce stand-alone programs that use the smallest amount of memory. When you create a stand-alone program, you can specify stub files on the LINK command line. The effect is to cancel or reduce support for a feature and thus reduce executable size.

For more information on stub files, see Chapter 15, "Optimizing Program Size and Speed," and Chapter 18, "Using LINK and LIB," in the *Programmer's Guide*.

Graphics Support Stub Files

The following list summarizes stub files that correspond to items in the second Specify Libraries and Run-Time Modules screen, Screen I/O and Graphics Support. Setup uses a designated stub file to build the BRT module if the corresponding menu item is turned *off*.

Turn this option *off*	To link with this stub file	Having this effect
Full Ctrl-Character Text I/O	TSCNIO*sm*.OBJ	Limits the program to text-only screen I/O with no support for special treatment of control characters. Removes support for all screen modes other than 0.
Graphic Screen Mode Support	NOGRAPH.OBJ	Removes support for all graphics statements and **SCREEN** modes other than 0.
CGA Graphics SCREEN Modes 1–2	NOCGA.OBJ	Removes support for **SCREEN** modes 1 and 2.
Hercules Graphics SCREEN Mode 3	NOHERC.OBJ	Removes support for **SCREEN** mode 3.
Olivetti Graphics SCREEN Mode 4	NOOGA.OBJ	Removes support for **SCREEN** mode 4.
EGA Graphics SCREEN Modes 7–10	NOEGA.OBJ	Removes support for **SCREEN** modes 7–10.
VGA Graphics SCREEN Modes 11–12	NOVGA.OBJ	Removes support for **SCREEN** modes 11–12.

Miscellaneous Support Stub Files

The following list summarizes stub files that correspond to items in the third Specify Libraries and Run-Time Modules screen:

Turn this option *off*	To link with this stub file	Having this effect
COM*n* Device OPEN and I/O	NOCOM.OBJ	Removes support for COM device filenames. Normally, LINK includes support for COM port communication when an **OPEN** statement uses a filename that is either a string variable or starts with "COM."

LPT*n* Device OPEN and I/O	NOLPT.OBJ	Removes support for LPT device filenames. Normally, LINK includes support for LPT port communication when an **OPEN** statement uses a filename that is either a string variable or starts with "LPT."
INPUT Floating-Point Values	NOFLTIN.OBJ	Allows programs to contain **INPUT**, **VAL**, and **READ** statements without linking in floating-point support. Use of this stub file assumes you will not input floating-point constants.
Transcendental Math	NOTRNEM*m*.LIB	Removes support for transcendental functions, which include trigonometric functions, exponential functions, and the following: the **CIRCLE** statement with a start or stop value, and the **DRAW** statement with A or T commands.
EMS Support for Overlays	NOEMS.OBJ	Prevents a program linked for overlays from using EMS memory. The program will be forced to swap to disk. This file is not built into the BRT module, but must be linked with your program's object files when creating the executable file.
EVENT Trapping	NOEVENT.OBJ	Removes support for **EVENT** trapping.
Full-Power INPUT Editor	NOEDIT.OBJ	Limits editing when a user enters data in response to an **INPUT** statement. The user will only be able to use Backspace to change entries.
Detailed Error Messages	SMALLERR.OBJ	Reduces length of run-time error messages.

Other Special Files Installed by Setup

In addition to stub files, Setup may install two other special files, depending on the configuration chosen. Both these files correspond to options in the third Specify Libraries and Run-Time Modules screen. As with stub files, Setup builds these files directly into the BRT module, but unlike stub files, you must turn the corresponding menu items *on* in order to link with the files. If you want to use them in a stand-alone program, include them on the command line when you link the program.

Turn this option *on*	To link with this file	Having this effect
Math Coprocessor Required	87.LIB	Provides most direct support for machines that have an 8087-family coprocessor installed. This option also removes software coprocessor emulation, so that programs with floating-point calculations can only run on computers with a coprocessor.
Overlays in DOS 2.1	OVLDOS21.OBJ	Required for a program with overlays to work under DOS 2.1.

Where to Go If You Need Additional Help

If you need additional help with any of the options in the Setup program, you can use the following table to locate information in the printed documentation. If you just need a quick overview of the new features in Microsoft BASIC, be sure to read Chapter 1, "New Features," if you haven't already done so.

For information on	See
ISAM	*Programmer's Guide*, Chapter 10, "Database Programming with ISAM"
Near and far strings	*Programmer's Guide*, Chapter 11, "Advanced String Storage"
Alternate and emulator math	*Programmer's Guide*, Chapter 15, "Optimizing Program Size and Speed," and Chapter 16, "Compiling with BC"
Overlays	*Programmer's Guide*, Chapter 18, "Using LINK and LIB"
Compiler options	*Programmer's Guide*, Chapter 16, "Compiling with BC"
Stub files	*Programmer's Guide*, Chapter 15, "Optimizing Program Size and Speed," and Chapter 18, "Using LINK and LIB"
Add-on libraries	*BASIC Language Reference*, Part 2
Toolboxes	*BASIC Language Reference*, Part 3, and *Programmer's Guide*, Chapter 6, "Presentation Graphics"

Chapter 3
Using QuickBASIC Extended

The QuickBASIC Extended (QBX) environment is a powerful tool for developing BASIC applications under DOS. This chapter introduces the QBX environment and covers the following information to get you started:

- Starting QBX and using the default editing and command keys.
- Creating and moving around in a program.
- Compiling programs from within QBX.
- Using symbol Help, watches, watchpoints, breakpoints, and CodeView support to locate and resolve errors in your program.
- Adding items to the menu bar and changing the default editing and command key assignments.
- Using the QBX command-line syntax.

Running QBX

To start QBX, type the following at the system prompt:

```
QBX
```

Figure 3.1 shows the QBX environment.

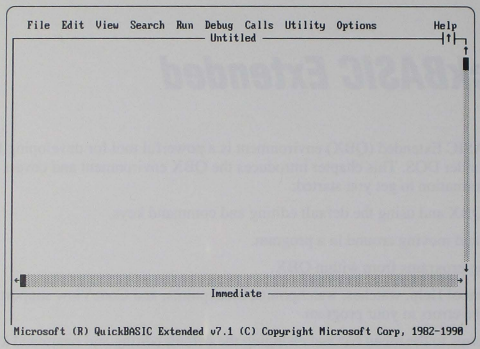

Figure 3.1. The QBX Environment

All environment tasks within QBX can be accomplished by using either the mouse or the keyboard. The following table lists commonly used key combinations by category and task.

Category	To do this	Press
Edit	Open a program	Alt+F, O
	Save a program	Alt+F, S
	Select text	Shift+direction key
	Copy selected text	Ctrl+Ins
	Cut selected text	Shift+Del
	Cut a single line	Ctrl+Y
	Paste copied or cut text	Shift+Ins
	Search for selected text	Ctrl+/
	Repeat last search	F3

Debug	Execute program	Shift+F5
	Execute to cursor position	F7
	Execute next statement	F8
	Execute next procedure	F10
	Continue execution	F5
	Add instant watch	Shift+F9
	Set/clear a breakpoint	F9
View/Windows	Next window	F6
	Previous window	Shift+F6
	View a SUB	F2
	View the next SUB	Shift+F2
	Display/hide output screen	F4
Help	Help on a topic	F1
	Help on using Help	Shift+F1
	Exit Help	Esc
	Redisplay last Help	Alt+F1

Writing a Program

After you start QBX, you can immediately begin writing a program. As you enter BASIC functions and statements, QBX interprets them and checks the syntax of each line. If you make a syntax error, QBX displays an error message dialog box and you must choose OK or press Esc to continue. QBX only checks for errors in syntax; to detect errors in logic, you must run your program. For example, type the following:

```
Now = TIME$
```

Press F5 to execute the program. QBX highlights **TIME$** and displays a "Type mismatch" error. Choose OK and correct the line to read:

```
Now$ = TIME$
```

Now the program will execute without an error.

QBX organizes your program by separating different procedures (**SUB**s and **FUNCTION**s). When you save your program, QBX automatically adds **DECLARE** statements as needed. For example, type the following:

```
SUB PrintTime
```

When you press Enter, QBX creates a new procedure with **SUB** and **END SUB** statements in the current window as follows:

```
SUB PrintTime

END SUB
```

To see the organization of the program, press F2. QBX displays the View SUBs dialog box. This dialog box lets you move between procedures and the main module of your program. Press Esc to return to the PrintTime procedure.

Add the following infinite loop between the **SUB** and **END SUB** statements:

```
DO
   CLS
   LOCATE 2, 1
   PRINT "The time is: "; TIME$;
   SLEEP 1
LOOP
```

Now choose Split from the View menu to open another window. Display the main module in the top window by typing Shift+F2 (View Next SUB). After the last statement in the main module, type:

```
CALL PrintTime
```

Run the program by pressing F5.

Press Ctrl+Break to interrupt the infinite loop and return to QBX. QBX highlights the last instruction executed (in this case, **LOOP**).

To save the program you just created:

1. From the File menu, choose Save.
2. Type the filename TIME.BAS, select Text Format, and press Enter.
   QBX saves the file as text. Text format files can be used by PWB, CodeView, and other tools. Binary format files can only be used by QBX and the BASIC Compiler.

Exit QBX by choosing Exit from the File menu. You can view the file you just saved by typing the following command from the command line:

```
TYPE TIME.BAS
```

The system displays the file as follows:

```
DECLARE SUB PrintTime ()
Now$ = TIME$
CALL PrintTime

SUB PrintTime
DO
   LOCATE 2
   PRINT "The time is: " + TIME$
   SLEEP 1
LOOP
END SUB
```

Notice that QBX adds a **DECLARE** statement and puts procedures after the main module in the file when it saves the file.

Other Statements Added by QBX

In addition to the **DECLARE** statement, QBX automatically adds a **DEF***type* statement to procedures under certain conditions. For example, if you start your program with a `DEFINT A-Z` statement, QBX adds a copy of that statement to the beginning of each new procedure you create.

If your program has existing procedures when you add a **DEF***type* statement to the main (calling) module, those procedures are not changed. Only new procedures get the automatic **DEF***type* statements.

Since the default data type in BASIC is single-precision, QBX will add a **DEFSNG** statement to any previously existing procedures that did not already have a **DEF***type* statement. These statements can only be seen by saving the file as text and viewing it outside QBX. For this reason, it is a good idea to start your program with a **DEF***type* statement.

Using the Immediate Window

The Immediate window at the bottom of the screen in QBX is a useful tool for executing a few lines of BASIC code without running the currently loaded program. Each line that you type in the Immediate window is executed when you press Enter. For example:

Run QBX. Press F6 to move the cursor to the Immediate window, and type:

```
CLS
PRINT "Today's date is: " + DATE$
```

Each time you press Enter at the end of a line, QBX executes that line of code and displays the output screen. You can cut lines from your program and paste them into the Immediate window. You can then scroll to the beginning of the pasted line and execute them one at a time by pressing Enter after each.

Lines typed in the Immediate window are not saved with your program. When you exit QBX, the Immediate window is cleared.

Using Quick Libraries

QBX uses special-format libraries called Quick libraries to make common routines available within the environment. To use a Quick library, you must specify it with the /L option when you start QBX. For example:

```
QBX /L CHRTBEFR.QLB
```

This command line starts QBX and loads the Presentation Graphics Quick library. The Presentation Graphics routines will be available to programs as you work in the environment.

For more information on Quick libraries, see Chapter 19, "Creating and Using Quick Libraries," in the *Programmer's Guide*.

Important Quick libraries created for QuickBASIC version 4.5 or earlier must be recreated from the original source code, and non-BASIC routines included in Quick libraries may need rewriting or relinking. For programming considerations when writing Quick libraries and instructions on creating Quick libraries, see Chapter 19, "Creating and Using Quick Libraries," in the *Programmer's Guide*.

Compiling a Program

To compile a program from QBX, load a BASIC source file and choose Make EXE from the Run menu. Figure 3.2 shows the Make EXE File dialog box that QBX displays.

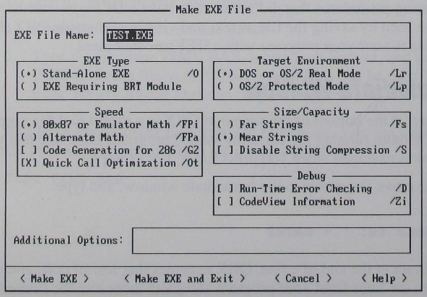

Figure 3.2 Make EXE File Dialog Box

By default, QBX uses the compiler options that cause your compiled program to run the way that it is interpreted within QBX. If a Quick library is loaded, QBX uses /O, /Ot, /Fs, /Lr, /FPi, /T, and /C:512 by default. If a Quick library is not loaded, QBX uses /O, /Ot, /Lr, /FPa, /T, and /C:512. For descriptions of the BASIC Compiler options, see online Help or Chapter 16, "Compiling with BC," in the *Programmer's Guide*.

To change these defaults, you can select any of the displayed compiler options or you can type additional options in the Additional Options field.

When you select Make EXE from the Make EXE File dialog box, QBX shells to DOS and runs the BASIC Compiler and LINK to build the executable file. If you have a Quick library loaded, QBX links your program with the object module library (.LIB) with the same base name as the loaded Quick library (.QLB).

If an error occurs when compiling or linking, the build halts and the error is displayed; you can press any key to return to QBX.

If the build is successful, you are returned to QBX.

Debugging a Program

To help locate errors in program logic, QBX provides the following features:

- Symbol Help
- Watches
- Watchpoints
- Breakpoints
- CodeView support

Symbol Help

When you ask for Help on a symbol, QBX displays a schematic of where and how that symbol is used in your program. Figure 3.3 shows the Help screen that is displayed when you ask for Help on the `Answer` symbol in the BOOKLOOK.BAS program.

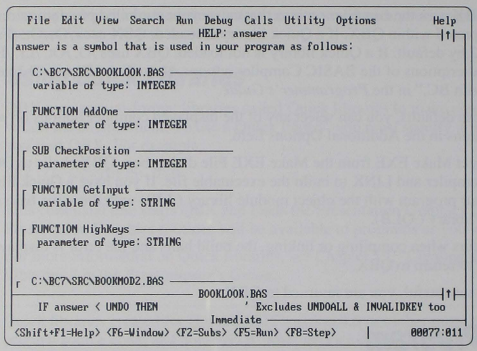

Figure 3.3 Symbol Help

Symbol Help is useful for resolving type mismatch errors and duplicate symbol names. Getting Help on a symbol makes it easy to determine a symbol's data type and the procedures in which it is used.

Watches

Watches let you observe the value of an expression as your program executes. This is especially helpful when your program yields unexpected results. Placing a watch on an expression and stepping through the program can help you locate where the problem is occurring.

To add a watch:

1. Select the expression you want to observe.
2. From the Debug menu, choose Add Watch.
   The expression's name and current value are displayed in a one-line window immediately below the menu bar.

As you run your program, QBX updates the value in the Watch window whenever the value of the expression changes. For example, set a watch on `N%` in the following lines of code:

```
FOR N% = 0 to 10 STEP 5
NEXT N%
```

Step through the procedure by pressing F8 five times. The value of `N%` displayed in the Watch window changes from 0 to 5 to 10.

You can set an instant watch to view the current value of an expression without adding a watch. An instant watch shows only the current value of the expression.

To set an instant watch:

1. Select the expression to view.
2. From the Debug menu, choose Instant Watch.
   The current value of the expression is displayed in a dialog box.
3. Choose Cancel to exit the dialog box without adding a watch or press Enter to add a watch for the expression.

To remove a watch, choose Delete Watch from the Debug menu, then select the watch to remove.

To remove all watches, choose Delete All Watch from the Debug menu.

Watchpoints and Breakpoints

"Watchpoints" are conditional statements that stop program execution when an expression is true (nonzero). "Breakpoints" are markers that stop program execution at a fixed location in the program. Watchpoints and breakpoints make it easier to get to a specific point in the logic or structure of the program while debugging.

To set a watchpoint:

1. Select the expression to watch.
2. From the Debug menu, choose Watchpoint.
   When you run the program by pressing F5, QBX will halt the program and take you to the current statement as soon as the value of the expression becomes nonzero.

To remove a watchpoint:

1. From the Debug menu, choose Delete Watch.
2. Select the watchpoint to remove and press Enter.

To set a breakpoint:

1. Place the cursor on a line of code where you want the program to stop executing.
2. From the Debug menu, choose Toggle Breakpoint.
   When you run the program by pressing F5, QBX will halt the program as soon as it reaches line of code with the breakpoint.

To remove a breakpoint, place the cursor on the line with the breakpoint and choose Toggle Breakpoint from the Debug menu.

CodeView Support

To use CodeView with programs created in QBX, you must save the source file in text format. CodeView cannot interpret files saved in QBX binary format.

After saving your program, compile it specifying the CodeView option (/Zi or /Zd). You can then load the file into CodeView from the command line. For information on getting started with CodeView, see Chapter 5, "Using CodeView." For detailed information about the CodeView environment and how expressions are evaluated in CodeView, see the CodeView online Help.

Customizing the Environment

You can add items to the menu bar and remap key assignments within the QBX environment.

Adding Items to the Menu Bar

To add an item to the menu bar:

1. From the Utility menu, choose Customize Menu.
2. Select Add from the Customize Menu dialog box.
3. Type the appropriate information in the displayed fields, and choose OK. For information about the field in the Customize Menu dialog box, press F1.

The new item will appear on the Utility menu.

Remapping Key Assignments

The default QBX editing and command keys can be modified by loading one of the following files with the /K: option when you start QBX:

Load this file	To change settings to those for this editor
ME.KEY	Microsoft Editor
BRIEF.KEY	BRIEF
EPSILON.KEY	Epsilon
QBX.KEY	QBX

For example, to change the default settings to those of the Microsoft Editor, start QBX with the following command line:

```
QBX /k:ME.KEY
```

When you change the default key settings, the change is saved in QBX.INI and those settings become the new default.

You can further customize these key assignments by using the MKKEY utility as follows:

1. Decode the key assignment file. For example:
   ```
   MKKEY -c ba -i QBX.KEY -o MYKEY.TXT
   ```
2. Edit the decoded file (in this case, MYKEY.TXT).
3. Encode the modified key assignment file. For example:
   ```
   MKKEY -c ab -i MYKEY.TXT -o MYKEY.KEY
   ```
4. Load the new key assignment file in QBX. For example:
   ```
   QBX /k:MYKEY.KEY
   ```

For more information on customizing key assignments, see the online Help topic "Configuring Keys" from the Help table of contents in QBX. For more information about MKKEY, see Appendix C in the *BASIC Language Reference*.

Controlling the Environment

You can control various aspects of the environment by using the QBX command-line options. The QBX command line has the following syntax:

QBX [[*options*]] *programname* [[/CMD *string*]]

Option	Short description
/AH	Allows dynamic arrays of records, fixed-length strings, and numeric data to be larger than 64K each.
/B	Displays QBX in black and white.
/C: *buffersize*	For use with the **OPEN COM** statement. Sets the size of the buffer receiving data with an asynchronous communications card. The default is /C:512.
/Ea	Allows arrays in expanded memory. Do not use /Ea with the /Es option.
/E:*n*	Limits the amount of expanded memory reserved for QBX use. If /E:*n* is not specified, all available expanded memory may be used.
/Es	Shares expanded memory between QBX Quick library and mixed-language routines that make use of expanded memory.
/G	Sets QBX to update a CGA screen as quickly as possible.
/H	Displays the highest resolution possible on your hardware.
/K:[[*keyfile*]]	Specifies a user-configurable key file to be loaded into QBX.
/L [[*libraryname*]]	Loads a Quick library. If *libraryname* is not specified, the default Quick library (QBX.QLB) is loaded.

/MBF	Causes the QBX conversion functions to treat IEEE-format numbers as Microsoft Binary format numbers.
/Nofrills	Makes additional memory available for program use by reducing the functionality of the environment.
/NOHI	Allows the use of a monitor that does not support high intensity. Not for use with COMPAQ laptop computers.
/CMD *string*	Passes string to the **COMMAND$** function. This option must be the last option on the line.

Complete descriptions of the QBX options are available through online Help or in Appendix C in the *BASIC Language Reference*.

Chapter 4
Using Programmer's WorkBench

The Programmer's WorkBench (PWB) is an advanced development environment that features a powerful editor and integrates the compiler, LINK, NMAKE, and the CodeView debugger. PWB is intended for tasks, such as developing under OS/2, that are not supported by QBX. Unlike QBX, PWB does not interpret BASIC code. This means that some features available in QBX, such as the Immediate window and executing a program line-by-line, are not available. In PWB, you must use CodeView to debug programs.

This chapter explains how to perform the following tasks:

- Starting PWB and using keyboard commands.
- Loading and saving a file, setting or clearing a program list, and using special features (such as marks and anchors) to help you write and modify your programs.
- Selecting build options and creating an executable file.
- Preparing a file for debugging, viewing compiler errors, using the source browser, and invoking CodeView.
- Changing the default settings and key assignments, creating macros, and reducing the time it takes PWB to load.
- Using the PWB command-line syntax.

Note Setup installs PWB only if you choose to install mixed-language tools from the Specify Files to Install menu during Setup. If you did not choose mixed-language tools and you wish to use PWB, you should run Setup again with the mixed-language tools option chosen.

Running PWB

To start PWB for the first time, type:

```
PWB
```

Figure 4.1 shows the PWB environment.

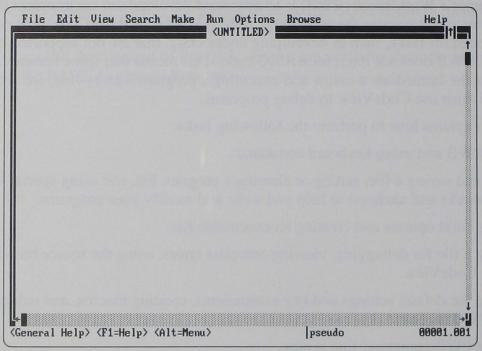

Figure 4.1 The PWB Environment

Important PWB includes extensive online Help. If you request Help and get an error message indicating that no Help files were open or that they could not be found, check your HELPFILES environment variable. HELPFILES can be set by running NEW-VARS.BAT (DOS) or NEW-VARS.CMD (OS/2) from the command line before starting PWB. HELPFILES can also be set from within PWB by choosing Environment from the Options menu. If you set HELPFILES from within PWB, that setting will not have an effect in CodeView.

Within PWB, environment tasks can be accomplished by using either the mouse or the keyboard. The following table lists commonly used default key combinations by category and task:

Category	To do this	Press
Edit	Select text	Shift+direction key
	Copy selected text	Ctrl+Ins
	Cut selected text	Shift+Del
	Cut a single line	Ctrl+Y
	Paste copied or cut text	Shift+Ins
	Undo previous edit	Alt+Backspace
	Redo edit	Ctrl+Backspace
	Search forward	F3
	Search backward	F4
File operations	Open a program	Alt+F, O
	Save a program	Shift+F2
	Exit (save changes)	Alt+F4
	Quit (abandon changes)	F9, F8
	Refresh (abandon changes)	Shift+F7
	Load previous file	F2
	Shell to system	Shift+F9
Debug/Browse	Next definition/reference	Ctrl+Num+
	Previous definition/reference	Ctrl+Num-
	Next build error	Shift+F3
	Previous build error	Shift+F4
Windows	Next window	F6
	Previous window	Shift+F6
	Resize a window	Ctrl+F8
	Maximize a window	Ctrl+F10

Help	Get Help on a topic	F1
	View table of contents	Shift+F1
	Exit Help	Esc
	Redisplay last Help	Alt+F1

Writing a Program

After you start PWB, you can immediately begin writing source code in the window that appears. PWB supports programming in BASIC, C, and other Microsoft languages. Unlike QBX, PWB does not interpret your program as you write it. Also unlike QBX, PWB does not check your BASIC syntax or automatically add **DECLARE** statements as you write your program. To check syntax in PWB, you must compile your program and view the compile results. For more information on how to do this, see the sections "Compiling and Linking a Program" and "Debugging a Program" later in this chapter.

Loading and Saving a File

To load a file in PWB, you can either type the filename on the command line when you start PWB or choose Open from the File menu within PWB.

Note If you are planning to use PWB in combination with QBX, you must save your source files in text format from QBX before you try to load them in PWB. PWB cannot use files saved in QBX binary format.

To save a file, choose Save from the File menu.

PWB maintains a history of the files you have opened, which is displayed at the bottom of the File menu. To return to a file displayed in this list, choose its name from the File menu. Figure 4.2 shows the File menu with several files in history.

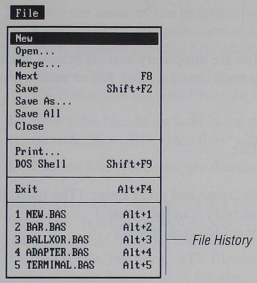

Figure 4.2 File History on the File Menu

When you edit a file and then exit PWB, the environment saves a list of the files you worked on in a special file named CURRENT.STS. PWB creates CURRENT.STS in the first directory indicated by your INIT environment variable. The next time you run PWB, it automatically opens the last file you worked on unless you explicitly specify a filename on the command line or use the /D or /DS options. For PWB command-line syntax and options, see the section "Controlling the Environment" later in this chapter.

Navigating in a File

PWB provides marks and anchors to help you navigate and edit large files. A "mark" is an invisible placeholder that you can set at a specified location in a file to go to that location quickly in the future. An "anchor" is a placeholder that determines the boundary of a block of selected text to cut, copy, or delete.

Setting and Using Marks

To set a mark:

1. Place the cursor at the location you want to mark, then choose Define Mark from the Search menu.
   PWB displays the Define Mark dialog box.
2. Type a name that you want to assign to the mark and press Enter. Note that spaces are not allowed in mark names.

To go to a mark:

1. From the Search menu, choose Goto Mark.
   PWB displays the Go To Mark dialog box.
2. Select the mark from the displayed list and choose OK.

Creating and Using a Mark File

Marks created as shown in the preceding section are temporary and are erased when you exit PWB. To create marks that can be reused, you must create a mark file or select an existing one before setting your marks.

To create a new mark file:

1. From the Search menu, choose Set Mark File.
   PWB displays the Set Mark File dialog box.
2. Type the name of the mark file you wish to create and press Enter. (The mark file should have an .MRK file extension to help you identify it in the future.)
   PWB displays a dialog box asking if you want to create the file.
3. Choose Yes.

To select an existing mark file:

1. From the Search menu, choose Set Mark File.
   PWB displays the Set Mark File dialog box.
2. Select a mark file from the displayed list and choose OK.

After you create a new mark file or select an existing one, it becomes the current mark file and any temporary marks are cleared. When you set new marks, they are added to the current mark file by default.

Setting and Using Anchors

Anchors help you select large blocks of text to cut, copy, or delete.

To set an anchor, place the cursor at the position you wish to anchor and choose Set Anchor from the Edit menu. You can only set one anchor at a time; moving the cursor and choosing Set Anchor again resets the anchor to the new position.

To select text between the current cursor position and the anchor, choose Select To Anchor from the Edit menu. PWB highlights the selected text.

Programming with Multiple Files

To work on a project containing multiple files, you must create a program list. A "program list" is a special PWB file that contains the build instructions and filenames for a project.

To create a program list:

1. From the Make menu, choose Set Program List.
   PWB displays the Set Program List dialog box.

2. Type the name of the program list you wish to create and press Enter. (The filename should have an .MAK file extension to help you identify it in the future.)
   PWB displays a dialog box asking you if you want to create the file.
3. Choose Yes.
   PWB displays the Edit Program List dialog box.
4. Select the files to include in the project from the displayed list. Do not select BASIC include files (.BI), since they will be included when the program is compiled.
5. Choose OK when you are done.

Build options are maintained in a state file that is maintained along with the .MAK file. A "state file" is a file containing information about the PWB editing session, such as current build options and file history. The state file maintained with a .MAK file has the same base name as the .MAK file, but has a .STS filename extension. When you set a program list, build options revert to those last saved in the state file. For information on setting build options, see the section "Compiling and Linking a Program" later in this chapter.

To use an NMAKE .MAK file in PWB, follow the procedure used to create a program list, but select the Use as a Non-PWB Makefile checkbox in the Set Program List dialog box before you press Enter in step 2.

Note PWB .MAK files are not compatible with QBX .MAK files and vice versa. If you are working on a project in both environments, you should create a different .MAK file for each environment.

Compiling and Linking a Program

Executable files can be created directly from the PWB environment. Options can be saved for debug and release versions of the executable file. All of the environment, compiler, and LINK options are set from the Options menu.

Changing Environment Variables

To change environment variables from those in effect when PWB was run:

1. From the Options menu, choose Environment.
   PWB displays the Environment Options dialog box.
2. Type the new environment variables in the fields provided. Press Tab to move to the next option, and press Enter when you are done.

The new environment variables take effect immediately and remain in effect until you exit PWB. To make permanent changes to environment variables, edit your AUTOEXEC.BAT file (DOS) or your CONFIG.SYS file (OS/2).

Setting Debug or Release Version

PWB maintains lists of debug and release options for compiling and linking a program. By default, debug options create an executable file for use with CodeView; release options create an optimized executable file without CodeView information. Setting release or debug version makes it easy to switch between the two sets of options quickly.

To set whether the executable file will be a debug version or a release version:

1. From the Options menu, choose Build Options.
   PWB displays the Build Options dialog box.
2. Select Debug or Release from the dialog box.
3. Choose OK.

Debug and release version options are set from the compiler and LINK options dialog boxes. To change compiler and LINK options for debug or release versions, see "Setting Compiler Options" and "Setting LINK Options" later in this chapter.

Setting Main Language and Initial Build Options

Since PWB is a mixed-language programming environment, many of the build options that are valid in some languages are not valid in others. PWB provides you with a way to set a main programming language and to create default build options based on the target environment. When working with BASIC, you should always leave Main Language set to BASIC. When working exclusively in another language, such as C, you will want to change the Main Language.

To change the Main Language:

1. From the Options menu, choose Build Options.
   PWB displays the Build Options dialog box.
2. Select Set Main Language.
   PWB displays the Set Main Language dialog box.
3. Select a language from the displayed list, and choose OK.
4. Select Set Initial Build Options box.
   PWB displays the Set Initial Build Options dialog box.
5. Select the line from the displayed list that best describes the executable file you want to build, and choose OK.
6. Choose OK again.

Three valid initial build options are provided for BASIC. You can save your current build options to create a new group of initial build options that you can use with future projects. To create a new group of initial build options for BASIC:

1. Set Main Language to BASIC (see the preceding procedure).

2. Set the build options. These include the settings for the debug and release versions of compiler, LINK, and Source Browser options.
3. From the Options menu, choose Build Options.
   PWB displays the Build Options dialog box.
4. Select Save Current Build Options.
   PWB displays the Save Current Build Options dialog box.
5. Type a name to identify your current build settings, and press Enter.
6. Choose OK.

PWB saves initial build options in your state file. (For more information on state files, see the section "Reducing Load Time" later in this chapter.) After initial build options are saved, they are available from the Set Initial Build Options dialog box.

Setting Compiler Options

To set the BASIC compiler options within PWB:

1. From the Options menu, choose BASIC Compiler Options.
   PWB displays the BASIC Compiler Options dialog box.
2. Select the appropriate displayed options, then select Set Debug Options or Set Release Options.
3. Select the appropriate options from the BASIC Compiler Debug/Release Options dialog box, and choose OK.
4. Type any additional options you wish to specify in the Additional Options field, and choose OK.

Note It is possible to set options in the Additional Options field that conflict with the options specified in the BASIC Compiler Options dialog box. Conflicting options will cause the compile to fail.

Setting LINK Options

To set LINK options within PWB:

1. From the Options menu, choose LINK Options.
   PWB displays the LINK Options dialog box.
2. Select the appropriate displayed options, then select Set Debug Options or Set Release Options.
3. Select the appropriate options from the LINK Debug/Release Options dialog box, and choose OK.
4. Type any additional options or libraries you wish to specify in the Additional Libraries and Additional Options fields, and choose OK.

Note Some LINK options, such as /INCREMENTAL and /TINY, are not valid for BASIC and will cause the build to fail. For a list of the valid LINK options and their uses, see Chapter 18, "Using LINK and LIB," in the *Programmer's Guide*.

Building an Executable File

PWB offers four choices from the Make menu for building an executable file. These selections allow you to:

- Compile or compile and link a single file.
- Compile changed files and relink with unchanged object files in a project.
- Compile and link all files (changed and unchanged) in a project.

For information on creating a project, see the section "Programming with Multiple Files" earlier in this chapter.

Compiling a Single File

Compiling a single file can be used as a first step in debugging a new program or as preparation for compiling and linking files in a program list.

To compile a single file:

1. Open the file to be compiled.
2. Set the compiler options.
3. From the Make menu, choose Compile File.
   PWB compiles the file. Any errors, are displayed in the Compile Results window.
4. PWB displays a dialog box confirming that the compile has completed. Select one of the buttons or press Esc to return to the active window.

Compiling and Linking a Single File

To compile and link a single-file program:

1. Open the file to compile and link.
2. Set build, compiler, and link options as needed.
3. From the Make Menu, choose Build or Rebuild All. The Build command recompiles the file if it has changed since it was last compiled, then links the file with the appropriate library to create an executable file. Rebuild All compiles the file whether or not it has changed, then links the file with the appropriate library to create an executable file.
   If the build fails, PWB displays compile or LINK errors in the Compile Results window.
4. PWB displays a dialog box confirming that the build has completed. Select one of the buttons or press Esc to return to the active window.

Compiling and Linking Files in a Project

PWB gives you two options for compiling and linking the files in a project: you can recompile only files that have changed and link with unchanged object files, or compile and link all files whether or not they have changed.

To compile and link files in a project:

1. Set or create a program list for the project (see "Programming with Multiple Files" earlier in this chapter).
2. Set build, compiler, and link options as needed.
3. From the Make Menu, select Build or Rebuild All. Build compiles only changed files in the project and links them with existing (unchanged) object files. Rebuild All compiles and links all files in the project.
   If the build fails, PWB displays compile or LINK errors in the Compile Results window.
4. PWB displays a dialog box confirming that the build has completed. Select one of the buttons or press Esc to return to the active window.

The preceding procedure uses the same compiler options for all files in the program list. To build a project using different compile options for a single file within that project:

1. From the Make menu, choose Clear Program List.
2. From the File menu, choose Open to open the file to compile.
3. Set the compiler options for the file.
4. From the Make menu, choose Compile.
5. From the Make menu, choose Set Program List and select the program list for the project.
6. From the Make Menu, choose Build.
   PWB does not recompile the file you compiled in step 4, since it has not changed. It will, however, recompile any changed files using the compile options last set for the program list, then link the objects with the appropriate library to create an executable file.

Debugging a Program

PWB provides the following diagnostic tools to help you locate errors, debug, and maintain your programs:

- Compile Results window
- Source browser
- CodeView support

Viewing Compiler Results

If errors occur when you compile or build a program, PWB displays the errors in the Compile Results window, as shown in Figure 4.3.

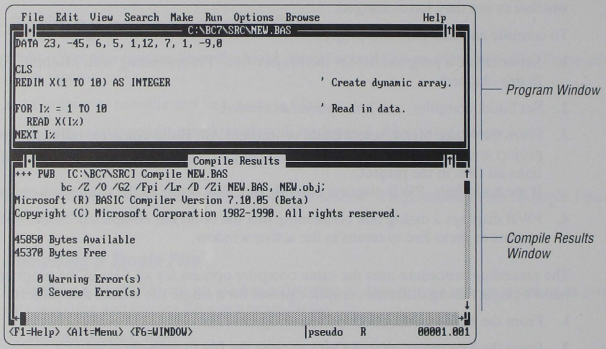

Figure 4.3 Compile Results Window

PWB automatically takes you to the location in your source file where the first error occurred. To go to the next error, press Shift+F3. To return to the previous error, press Shift+F4.

To close the Compiler Results window, choose Compile Results from the View menu.

Using the Source Browser

The PWB Source Browser provides a way to view relationships within a program and to quickly locate specific procedures, variables, and constants.

In order to use the Source Browser, it must be turned on before compiling and linking. Once the Source Browser is on and the program has been compiled and linked, the selections on the Browse menu become active. The Browse menu is shown in Figure 4.4.

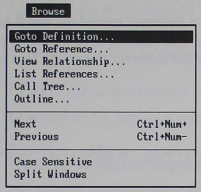

Figure 4.4 Browse Menu Selections

Browser Terminology

Since the Source Browser can be used with programs written in languages other than BASIC, some of the terms the Browser uses to refer to symbols may not be familiar. Keep in mind the following equivalents:

Source Browser term	BASIC equivalent
Functions	Procedures (**SUB**s and **FUNCTION**s)
Types	User-defined types
Macros	Constants

Turning On the Source Browser

To generate information for the PWB Source Browser, you must turn on browse information and compile the program:

1. From the Options menu, choose Browse Options.
   PWB displays the Browse Options dialog box.
2. Select Generate Browse Information and any additional options.
3. Choose OK.
4. Compile and link the program.

By default, PWB creates full browser information by compiling with the /FBx option. The /FBx option generates browser information for global and local definitions. To generate information for global definitions only, thus reducing the size of the resulting Browser files, you can perform the following steps before compiling and linking:

1. From the Options menu, choose BASIC Compiler Options.
   PWB displays the BASIC Compiler Options dialog box.
2. Select Set Debug Options or Set Debug/Release Options.
3. From the Set Options dialog box, choose Restricted Browse Info /FBr, and choose OK.
4. Choose OK again.

When you compile and link from PWB with the Source Browser turned on, the BASIC compiler generates a Browser information file with an .SBR extension. PWB then uses PWBRMAKE.EXE to convert this file into one which the Browser can read. This new file has a .BSC file extension.

Locating Program Symbols

The Source Browser helps you quickly find where procedures, variables, user-defined types, and constants are defined and referred to.

To locate a definition:

1. From the Browse menu, choose Goto Definition.
   PWB displays the Goto Definition dialog box as shown in Figure 4.5.
2. Select the item you wish to locate from the Names list box.
   PWB displays the locations where the definition occurs in the Defined in: list box.
3. Select the occurrence of the definition you wish to go to from the Defined in: list box.
4. Choose OK.
   PWB takes you to the location in the file where the selected definition occurs.

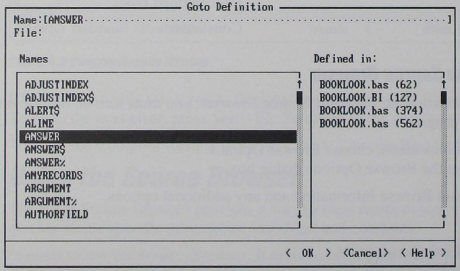

Figure 4.5 The Goto Definition Dialog Box

To locate a reference:

1. From the Browse menu, choose Goto Reference.
   PWB displays the Goto Reference dialog box.
2. Select the item to locate from the Names list box.
   PWB displays the locations where the reference occurs in the Referenced in: list box.

3. Select the occurrence of the reference you wish to go to from the Referenced in: list box.
4. Choose OK.
   PWB takes you to the location in the file where the selected reference occurs.

To go to the next reference or definition of a symbol, press Ctrl+Keypad Plus Sign (+).

To return to the previous reference or definition of a symbol, press Ctrl+Keypad Minus Sign (-)

Viewing Relationships Between Symbols

The Source Browser helps you view relationships between procedures, variables, user-defined types, and constants. These relationships include where a symbol is defined and used, where a procedure is called, what calls are made by a procedure or module, etc.

To view how a single symbol relates to the rest of the program:

1. From the Browse menu, choose View Relationship.
   PWB displays the View Relationship dialog box, as shown in Figure 4.6.
2. Select Program Symbols from the Operation list box.
3. Select the symbol you want to view from the list box.
4. Select the operation you want to perform from the Operation list box.
5. Choose OK.

To redisplay all the symbols, select Program Symbols from the Operation list box and choose OK.

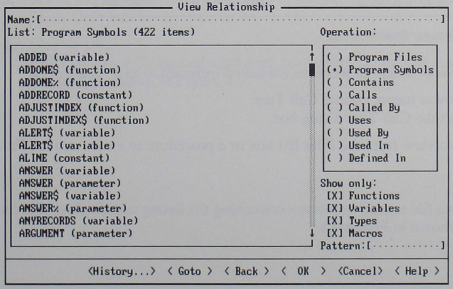

Figure 4.6 View Relationship Dialog Box

To view a listing of the different types of symbols and where they are used:

1. From the Browse menu, choose List References.
   PWB displays the List References dialog box.
2. Select the types of symbols to list.
3. Choose OK.
   PWB creates a file named <browse> containing the call tree and displays it in the active window, as shown in Figure 4.7.

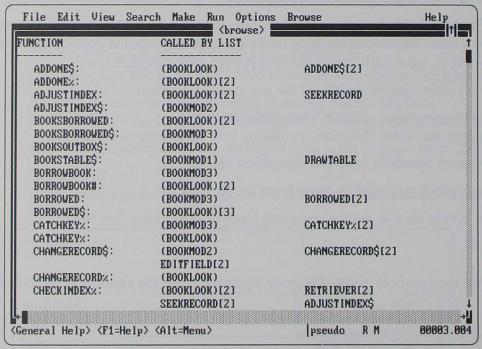

Figure 4.7 Source Browser Reference List

To view an outline of where procedures are called (referred to as a "call tree"):

1. From the Browse menu, choose Call Tree.
   PWB displays the Call Tree dialog box.
2. Select a file to view from the Files list box or a procedure to view from the Functions list box.
3. Choose OK.
   PWB creates a file named <browse> containing the listing and displays it in the active window, as shown in Figure 4.8.

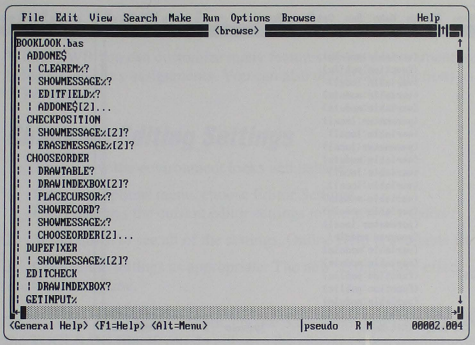

Figure 4.8 Source Browser Call Tree

To view an outline of a program:

1. From the Browse menu, choose Outline.
   PWB displays the Outline dialog box.
2. Select the file to view from the File List list box.
3. Select the type of symbols you want to include in the outline from the options under the Show Only label.
4. Choose OK.
   PWB creates a file named <browse> containing the outline and displays it in the active window, as shown in Figure 4.9.

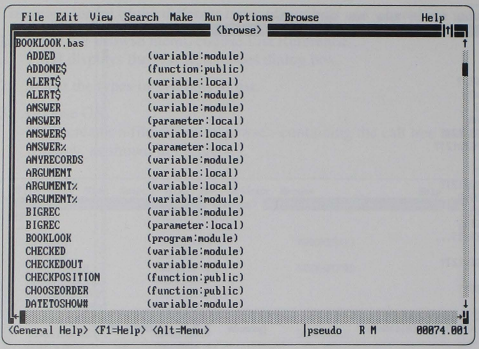

Figure 4.9 Source Browser Program Outline

Hint

By default, the <browse> file replaces the current program file in the active window. You can set PWB to create a new window for the <browse> file by choosing Split Window from the Browse menu before viewing symbol relationships.

Using CodeView

To start CodeView from PWB:

1. Build an executable file using the CodeView options for the compiler (/Zi or /Zd) and linker (/CO). These are set by default if you are building a debug version of the program. See the section "Setting Debug or Release Version" earlier in this chapter.
2. From the Options menu, choose CodeView Options to set the CodeView command-line options.
3. From the Run menu, choose Debug.
   PWB exits and loads CodeView.

When you exit CodeView, PWB is restarted.

For more information on using CodeView to debug your programs, see Chapter 5, "Using CodeView."

Customizing the Environment

Within PWB you can customize many features of the environment, such as screen colors, editor behavior, and key assignments. You can also define new functions as macros and assign them to keystrokes.

Changing Editing Settings

To change how the environment looks and behaves:

1. From the Options menu, choose Editor Settings.
   PWB displays the current editor settings in the active window.
2. Scroll down to see all of the settings. Online Help is available for each of the settings.
3. Change the settings as appropriate. The new settings take effect when you move the cursor to the next line.

To make the changes permanent, choose Save from the File menu. PWB saves the changes to the TOOLS.INI file. If you do not save the changes, they are discarded when you exit PWB.

To return to the current file, press F2.

Assigning Keystrokes

To change key assignments or to assign new functions to keys:

1. From the Options menu, choose Key Assignments.
   PWB displays the current key assignments in the active window.
2. Scroll down to see all of the assignments. Online Help is available for each of the editor functions.
3. Change the key assignments as appropriate. The new assignments take effect when you move the cursor to the next line.

To make the changes permanent, choose Save from the File menu. PWB saves the changes to the TOOLS.INI file. If you do not save the changes, they are discarded when you exit PWB.

To return to the current file, press F2.

The key assignments are divided into sections for each part of PWB. A label at the beginning of each section indicates where the assignments are used. For example, the [pwb] tag indicates that the subsequent assignments apply to the entire PWB environment; the [pwb-pwbbasic] tag indicates assignments that apply only to the BASIC Compiler selection from the Options menu; and the [pwb-pwbhelp] tag indicates settings that apply to the online Help system.

A table of key assignments unused by PWB is included after the [pwb-pwbutils] tag. These key combinations may be used for currently unassigned editor functions or for new functions created as macros.

Creating Macros

To record a macro:

1. From the Edit menu, choose Record On.
   PWB displays the Set Macro Record dialog box.
2. Type the name you want to assign to the macro in the Name field.
3. Tab to the Key Assignment field and type the names of the keystrokes you want to assign to the macro. If you place the cursor between the braces, { }, and press the key sequence you want to assign to the macro, the correct description of the key sequence is displayed in the field to the right.
4. Choose OK.
5. Perform the actions you want to record.
6. Choose Record On again to end the recording.

To make the recorded macro permanent:

1. From the Edit menu, choose Edit Macro.
   PWB displays the currently defined macros in the active window.
2. From the File menu, choose Save.
3. Press F2 to return to the current file.

When you make a macro permanent, PWB writes the macro definition and key assignment to your TOOLS.INI file. To modify the macro after you have made it permanent, you must edit the TOOLS.INI file.

Examining macro definitions in TOOLS.INI is a good way to learn the syntax of macros. Example macros are included in the TOOLS.PRE file included with BASIC. Online Help is also available on how to create and modify macros in PWB.

Reducing Load Time

When you start PWB, it automatically reads your TOOLS.INI file and a state file named CURRENT.STS from the directory defined in your INIT environment variable. Together, these files control the editor settings, macro definitions, and key assignments used by PWB. Since reading these files takes time, you can reduce load time by:

- Keeping the PWB entries in TOOLS.INI as short as possible.
- Keeping CURRENT.STS as small as possible.
- Not reading one or both of these files when you start PWB.

PWB also automatically loads any extension files it finds whose filenames start with the letters "PWB" in the directory from which you start PWB. These extension files make certain functions, such as the BASIC Compiler options, available in the environment. You can further reduce load time by disabling this auto-load feature and explicitly loading only the extensions you need.

Minimizing TOOLS.INI

The entries for PWB in your TOOLS.INI file appear after the [pwb] label in that file. TOOLS.INI can be edited and saved in the same way as any other file. Macros and settings that you may not use can be commented out by placing a semicolon (;) as the first character on the line. Lines that have been commented out are not read when PWB starts and will reduce load time slightly.

Minimizing CURRENT.STS

PWB creates the CURRENT.STS file to save file history and some editor settings (such as build options and file history). This file can be kept small by reducing the number of files you keep in history and by keeping the default editor settings. To reduce the number of files PWB keeps in history, you can reduce the number assigned to the tmpsav switch after the [pwb] label in the TOOLS.INI file. Reducing this number reduces the number of files that can be recalled from the File menu without using the Open File dialog box.

Ignoring TOOLS.INI and CURRENT.STS

You can prevent PWB from reading TOOLS.INI or CURRENT.STS or both files by using one of the /D options when you start PWB from the command line. The /DS option prevents CURRENT.STS from being read. The /DT option prevents TOOLS.INI from being read. The /DST option prevents both files from being read. Although these options prevent the information in these files from being available to PWB, they can be useful for some programming tasks.

Starting PWB with /DS is especially useful when working with a program list, since PWB creates a startup file to save the editor settings for each program list you create. When you set a program list, PWB automatically reads the new editor settings from the startup file for the program list. This file has the same base name as your .MAK file, but has an .STS filename extension. For more information on PWB command-line options, see the section "Controlling the Environment" later in this chapter.

Loading PWB Extensions Explicitly

PWB does not automatically load its extensions if you include the /DA option when you start PWB from the command line. This significantly reduces the time it takes to start PWB, but also limits the features that are available in the environment. You can restore features by explicitly loading extensions through settings in the TOOLS.INI file. For example:

```
[pwb - .BAS .BI]
   LOAD: C:\BC7\BIN\PWBBASIC.MXT;C:\BC7\BIN\PWBUTILS.MXT
```

These lines explicitly load the BASIC and utilities PWB extensions when you edit a file with the .BAS or .BI filename extension. If these lines exist in your TOOLS.INI file and you start PWB with the following command line, only the BASIC functions are available:

```
PWB /DA test.bas
```

The following table lists the filenames of the PWB extensions and describes the functions they provide:

Real-mode file	Protected-mode file	Provides support for
PWBBASIC.MXT	PWBBASIC.PXT	BASIC Compiler functions.
PWBROWSE.MXT	PWBROWSE.PXT	Source Browser functions.
PWBC.MXT	PWBC.PXT	C Compiler functions. This file is included on the BASIC distribution disks, but is not installed by Setup.
PWBHELP.MXT	PWBHELP.PXT	Online Help functions.
PWBUTILS.MXT	PWBUTILS.PXT	LINK, NMAKE, CodeView, and build functions.

Controlling the Environment

The full syntax for starting PWB from the command line is as follows:

PWB [[*options*]] [[*programname*]] [[*programname*]] ...

Unlike QBX, *programname* must include its filename extension if one exists. If you specify more than one *programname*, the first name you specify is the first file PWB edits. The remaining files are loaded if you select choose Next from the File menu.

Option	Action
/e *string*	Executes the PWB commands specified in *string* when PWB starts up.
/t *file*	Indicates that the specified *file* is temporary and should not be kept in file history. You must precede the name of each temporary file with a /t.

/D[[A | S | T]]	Prevents PWB initialization files from being read at startup. The /DA option prevents extensions from being loaded automatically. The /DS option prevents the CURRENT.STS file from being read. The /DT option prevents the TOOLS.INI file from being read. The /D is equivalent to /DAST.
/P[[F *file* | L | P *file*]]	Loads a program list. The /PF option loads the NMAKE .MAK file specified by *file*. The /PL option loads the last .MAK file used in PWB. The /PP option loads the PWB .MAK file specified by *file*.
/r	Specifies that all files opened are read-only.
/m *position*	Puts the cursor at the file location specified by *position*. For example: `PWB /m 120.50 BOOKLOOK.BAS` The preceding command line loads the file BOOKLOOK.BAS and places the cursor at line 120, character position 50.
/?	Lists the command-line options for running PWB.

Chapter 5
Using CodeView

The Microsoft CodeView debugger helps you locate, identify, and resolve bugs in programs under DOS or OS/2. CodeView can help you debug programs compiled from within QBX or PWB, or from the command line. CodeView can be accessed directly from the Programmer's WorkBench (PWB). Together, PWB and CodeView provide a complete development system for mixed-language and OS/2 program development.

This chapter describes:

- Preparing your programs for debugging with CodeView.
- Running CodeView.
- Debugging a program.
- Controlling the flow of execution while debugging.
- Displaying and changing variables.
- Advanced CodeView debugging techniques.
- Customizing CodeView using the TOOLS.INI file.
- Controlling CodeView with command-line options.

Preparing BASIC Files for CodeView

CodeView can debug programs created with any editor or environment. QBX and PWB provide CodeView options for compiling within the environment. With QBX, however, you must save your source file in text format before compiling with the CodeView option. Source files saved in QBX binary format are not compatible with CodeView.

Programming Style

To make it easier to debug programs in CodeView, you should avoid the following programming practices:

- Using the colon (:) to place multiple BASIC statements on a single line.

  CodeView can step through a program one line at a time. If more than one statement occurs on a line, CodeView treats the line as a single statement.

- Placing executable statements in include files.

  You cannot trace through executable statements that occur in an include file. Placing executable statements in include files is not a good programming practice.

- Using non-unique array names.

  Arrays should be named uniquely. If a variable has the same name as an array in your program, CodeView assumes that symbol is a variable, not an array. You will not be able to perform some operations on that array from the CodeView Command window.

Compiling and Linking

To debug a program in CodeView, you must compile the program with the /Zi or /Zd options and link using the /CO option. The /Zi option makes the executable file larger. You can reduce the size of programs by compiling with /Zd. You will not be able to view symbols in the Local window within CodeView, however.

You should not use the optimize (/Ot) option when compiling a program for use with CodeView. Although CodeView can debug an executable file compiled with /Ot, you will not be able to pause program execution on lines containing **SUB**, **FUNCTION**, or **DEF FN** statements.

Object files compiled for use with CodeView must be linked using the /CO option. When linking in DOS, you cannot use the /EXEPACK and /CO together. If your program needs to be packed to fit in memory, use the CVPACK utility to compress the executable file for debugging.

The syntax for CVPACK is:

CVPACK [[/HELP | /P]] *filename*

If you specify the /P option, CVPACK rewrites the file as compactly as possible. If you don't specify the /P, CVPACK rewrites the file as quickly as possible. In either case, the program takes up the same amount of memory.

Running CodeView

To start CodeView from the command line, use the CV real-mode or CVP protected-mode command and specify an executable file. For example:

```
CV TEST.EXE
```

To start CodeView from PWB in real or protected mode, choose Debug from the Run menu. By default, CodeView displays three windows, as shown in Figure 5.1.

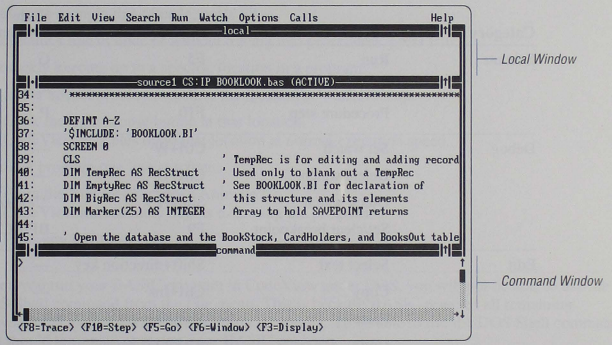

Figure 5.1 The CodeView Debugger

The "Local window" displays the current symbols along with their addresses, types, and values. This window is empty until you begin executing your program.

The "Source window" displays the program's source code. Since the program loaded in CodeView is a compiled executable program, you cannot make changes to the source from within CodeView. The Source window shows you where you are in a program and lets you execute the program statement-by-statement.

The "Command window" lets you enter CodeView commands and display and evaluate BASIC expressions. Although you cannot make changes to a program from CodeView, you can use the Command window to change the value of variables, write data to memory, and perform a variety of other tasks. CodeView supports a limited number of BASIC intrinsic functions which can be called from the Command window. See the CodeView online Help on BASIC intrinsic functions for more information.

Important CodeView includes extensive online Help. If you request Help and get an error message indicating that no Help files were open or that they could not be found, check your HELPFILES environment variable. HELPFILES can be set by running NEW-VARS.BAT (DOS) or NEW-VARS.CMD (OS/2) from the command line before starting CodeView. Setting HELPFILES in the PWB environment has no effect in CodeView.

Within CodeView, environment tasks can be performed by using either the mouse or the keyboard. The following table lists commonly used key combinations by category and task. The Command window equivalent is given where one exists. (See the online Help on CodeView commands for a complete list and description of CodeView commands.)

Category	Task	Key sequence	CodeView command
Execute	Run	F5	G
	Step single statement	F8	T
	Procedure step	F10	P
Debug	Set watch	Ctrl+W	—
	Delete watch	Ctrl+U	—
	Quick watch	Shift+F9	??*expression*
	Set/clear breakpoint	F9	BP/BC
Edit	Select text	Shift+direction key	—
	Copy	Ctrl+Ins	—
	Paste	Shift+Ins	—
	Undo	Alt+Backspace	—
Windows	Next window	F6	—
	Previous window	Shift+F6	—
	Resize a window	Ctrl+F8	—
	Maximize a window	Ctrl+F10	—
Help	Get Help on a topic	F1	H
	View table of contents	Shift+F1	—
	Exit Help	Esc	—
	Redisplay last Help	Alt+F1	—

Note The Edit tasks are used to copy and paste expressions, values, and blocks of memory. You cannot Paste into the Source window.

Debugging Your Program

CodeView is similar to QuickBASIC Extended (QBX) in that you can watch the value of variables while you execute parts of your program. Like QBX, you can execute one line at a time, one procedure at a time, or up to a specific location. Unlike QBX, you cannot rewrite your program within CodeView.

CodeView also lets you directly manipulate the values stored in variables and in memory.

Running Your Program

To execute a single line of code, press F8.

To execute a line of code or without tracing into procedures, press F10.

To quickly execute up to a specific location in a program:

1. In the Source window, scroll to the location at which to stop.
2. Click the right mouse button at that location.
   CodeView executes up to that location at normal execution speed.

To run a program one line at a time:

1. From the Run menu, choose Animate.
   CodeView immediately begins executing the program.
2. Press any key to stop.

Important When you run your BASIC program in CodeView under DOS, you will not be able to use the DOS Shell command from the File menu. This is because BASIC reserves all remaining conventional memory while the program is running. To be able to use the DOS Shell command you should terminate or restart your BASIC program.

Viewing Current Data

When you start executing a program in CodeView, the debugger automatically displays all current variables in the Local window. The Local window shows the address, data type, name, and value of each variable in the current procedure. For example:

```
2FCE:0062  STRING          FULLNAME$ = "Dan Webb"
```

The value 2FCE:0062 is the address of the variable in memory, STRING is the data type, FULLNAME$ is the variable name, and "Dan Webb" is the current value of the variable.

For arrays and user-defined data types, CodeView displays a summary line. For example:

```
+2FCE:0070 INTEGER    AZ%[array] = ?CANNOT DISPLAY
+2FCE:004a type  TESTRECOR FILEBUFFER = {...}
```

The plus sign (+) to the left of the memory address shows that there is more information. To see the elements in the structure TESTRECOR, select the TESTRECOR line. The line expands as follows:

```
-2FCE:0070 type     TESTRECOR FILEBUFFER
    STRING * 20     NAMEFIELD = "Jeff Quayle"
    SINGLE *        SCOREFIELD = "1370"
```

The minus sign (-) to the left of the memory address shows that the line is fully expanded.

You cannot expand BASIC arrays. Instead, you must set watches on elements in the array to see their values.

Using Watches

Watches let you observe the value of a variable or expression as your program executes. Since CodeView displays the value of local variables in the Local window, CodeView's watches are mainly useful when the Local window is not displayed or for observing array elements and expressions.

To add a variable to the Watch window:

1. Place the cursor on the variable and press Ctrl+W.
   CodeView displays the Add Watch dialog box.
2. Choose OK.
   CodeView adds the variable to the Watch window, as shown in Figure 5.2.

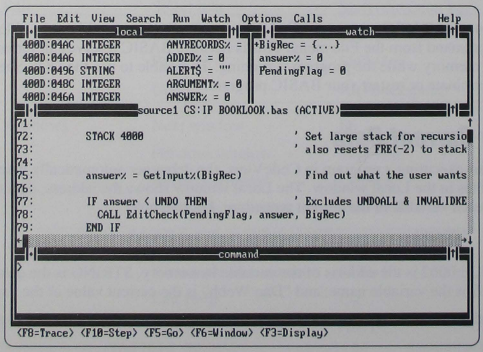

Figure 5.2 CodeView Watch Window

A variable may be followed by the message "Watch Expression Not in Context." This message appears when program execution has not yet reached the statement in which the variable is defined. Global variables, which can be watched from anywhere in the program, never cause CodeView to display this message.

To remove a variable from the Watch window, place the cursor on any line in the Watch window and press Ctrl+Y to delete the line.

You can place as many variables as you like in the Watch window; the quantity is limited only by available memory. CodeView automatically updates all watched variables as the program runs. This can slow execution speed considerably.

Displaying Expressions in the Watch Window

The Add Watch dialog box prompts for an expression, not simply a variable name. As this suggests, you can enter an expression (that is, any valid combination of variables, constants, and operators) for CodeView to evaluate and display.

You are not limited to evaluating BASIC expressions. The Language command from the Options menu offers a choice of BASIC, C, or FORTRAN evaluation for all expressions. The ability to select the language evaluator is especially useful when debugging mixed-language programs.

By reducing several variables to a single, easily read value, an expression can be easier to interpret than the components that comprise it.

Displaying Array Elements Dynamically

You can display a single element of an array using a specific subscript. You can also specify a variable array element, which changes as some other variable changes. For example, suppose that the loop variable `p` is a subscript for the array `CatalogPrice@`. The Watch window expression `CatalogPrice@(p)` displays only the array element currently specified by `p`, not the entire array.

You can mix constant and variable subscripts. For example, the expression `BigArray%(3,i)` displays the element in the third row of the array to which the index variable `i` points.

Using Quick Watch

A quick watch shows the current value of a variable. This is useful for quickly finding the value of a variable at some point in the program without adding it to the Watch window.

To display a quick watch:

1. Place the cursor on the variable to watch.
2. From the Watch menu, choose Quick Watch (or press Shift+F9).
   CodeView displays the Quick Watch dialog box with the value of the variable.

The Quick Watch display automatically expands user-defined types to their first level. You can expand or contract an element just as you would in the Watch window by placing the cursor on the appropriate line and pressing Enter.

Displaying Memory

You can view data at specific memory addresses using CodeView.

To display memory, choose Memory from the View menu. CodeView displays the Memory window, as shown in Figure 5.3.

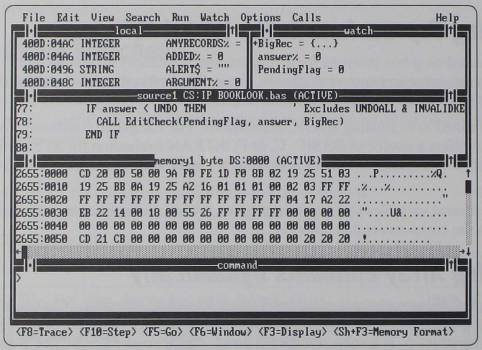

Figure 5.3 CodeView Memory Window

Up to two Memory windows can be open at once.

By default, memory is displayed as hexadecimal byte values, with 16 bytes per line. At the end of each line is a second display of the same memory in ASCII form. Values that correspond to printable ASCII characters (decimal 32 through 127) are displayed in that form. Values outside this range are shown as periods.

Byte values are not always the most convenient way to view memory. If the area of memory you're examining contains character strings or floating-point values, you might prefer to view them in a directly readable form.

To change the way data is displayed in the current Memory window:

1. From the Options menu, choose Memory Window.
   CodeView displays the Memory Window Options dialog box.
2. Select the format for display, and choose OK.

You can also directly cycle through these display formats by pressing F3.

If a section of memory cannot be displayed as a valid floating-point number, the number shown includes the characters NAN (not a number).

Displaying the Processor Registers

To view processor registers, choose Register from the View menu. CodeView opens a window on the right side of the screen, as shown in Figure 5.4. The current values of the microprocessor's registers appear in this window.

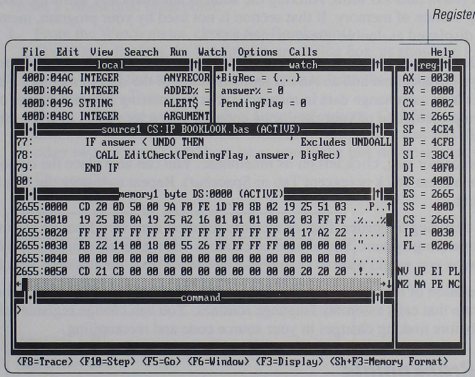

Figure 5.4 CodeView Register Window

At the bottom of the window is a group of mnemonics representing the processor flags. When you first open the Register window, all values are shown in normal-intensity video. Any subsequent changes are marked in high-intensity video. For example, suppose the overflow flag is not set when the Register window is first opened. The corresponding mnemonic is NV and appears in light gray. If the overflow flag is subsequently set, the mnemonic changes to OV and appears in bright white.

Choosing the 386 Instructions command (DOS only) from the Options menu displays the registers as 32-bit values if your computer uses an 80386 processor. Selecting this command a second time reverts to a 16-bit display. This command is not available in protected mode; the registers are shown in 16-bit display only.

You can also display the registers of an 8087/287/387 coprocessor in a separate window by choosing the 8087 command from the View menu. If your program uses the coprocessor emulator, the emulated registers are displayed instead.

Modifying the Values of Variables, Registers, and Memory

You can easily change the values of numeric variables, registers, or memory locations displayed in the Watch, Local, Memory, Register, or 8087 window. Simply place the cursor at the value you want to change and edit it to the appropriate value. To undo the last change you made, press Alt+Backspace. You cannot directly change the value of string variables.

The starting address of each line of memory displayed is shown at the left of the Memory window, in *segment:address* form. Altering the address automatically shifts the display to the corresponding section of memory. If that section is not used by your program, memory locations are displayed as double question marks (??).

When you select Byte display from the Memory Window Options dialog box, CodeView presents both a hexadecimal and an ASCII representation of the data in memory. (Byte display is the default.) You can change data in memory either by entering new hex values over the hexadecimal representation of your data or by entering character values over the character representation.

To toggle a processor flag, click left on its mnemonic. You can also place the cursor on a mnemonic, then press any key (except Tab or Spacebar). Repeat to restore the flag to its previous setting.

The effect of changing a register, flag, or memory location may vary from no effect at all to crashing the operating system. You should be cautious when altering *machine-level* values; most of the items you would want to change can be altered from the Local or Watch window.

Direct manipulation of register values can be valuable, however, when you are debugging a BASIC program that calls assembly language routines. You can change register values to test assumptions before making changes in your source code and recompiling.

Setting Breakpoints

You can skip over the parts of the program that you don't want to examine by specifying one or more lines as breakpoints. When you start the program by pressing F5, the program executes at full speed up to the first breakpoint, then pauses. Pressing F5 continues program execution up to the next breakpoint, and so on.

You can set as many breakpoints as you like (limited only by available memory). There are two ways to set breakpoints:

- Double-click anywhere on the desired breakpoint line. The selected line is highlighted to show that it is a breakpoint. (CodeView highlights lines that have been selected as breakpoints.) To remove the breakpoint, double-click on the line a second time.
- Place the cursor anywhere on the line at which you want execution to pause. Press F9 to select the line as a breakpoint. Press F9 a second time to remove the breakpoint.

A breakpoint line must be a program line that represents executable code. You cannot select a blank line, a comment line, or a declaration line (such as a variable declaration) as a breakpoint.

A breakpoint can also be set at a function or an explicit address. To set a breakpoint at a function, enter its name in the Set Breakpoint dialog box. To set a breakpoint at an address, enter the address in *segment:offset* form.

Once execution has paused, you can continue execution by pressing F5.

Setting Conditional Breakpoints

CodeView lets you set conditional statements that stop execution when an expression becomes true or changes value. (QBX refers to these conditional breakpoints as "watchpoints.")

To set a conditional breakpoint:

1. From the Watch menu, choose Set Breakpoint.
   CodeView displays the Set Breakpoint dialog box, as shown in Figure 5.5.
2. Select one of the conditional break options from the dialog box, such as Break When Expression is True.
   If you select one of the Break at Location options, CodeView takes you to the line number displayed in the Location field when the breakpoint is reached. If you select one of the other two options, CodeView takes you to the last statement that was executed.
3. Choose OK.

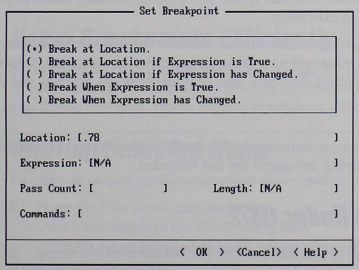

Figure 5.5 CodeView Set Breakpoint Dialog Box

Note

When a breakpoint is tied to a variable, CodeView must check the variable's value after each machine instruction is executed. This slows execution greatly. For maximum speed when debugging, either tie conditional breakpoints to specific lines, or set conditional breakpoints only after you have reached the section of code that needs to be debugged.

Replaying a Debug Session

CodeView can automatically create a history file with all the debugging instructions and input data you entered when testing a program. This history file is used to replay the debug session to a specific point.

To record a debug session, choose History On from the Run menu.

To stop recording, choose History On a second time.

Recordings can be used to keep track of debugging. You can quit after a long debugging session, then pick up the session later in the same place.

The principal use of recording a debug session is to allow you to back up when you make an error or overshoot the section of code with the bug. For example, you may have to execute a function manually many times before its bug appears. If you then enter a command that alters the machine's or program's status and thereby lose the information you need to find the cause of the bug, you have to restart the program and manually repeat every debugging step to return to that point. Even worse, if you don't remember the exact sequence of events that exposed the bug, it could take hours to find your way back.

Recording your session eliminates this problem. Choosing Undo from the Run menu automatically restarts the program and rapidly executes every debug command up to (but not including) the last one you entered. You can repeat this process as many times as you like until you return to the desired point in execution.

To add steps to an existing recorded session, choose History On, then select Replay. When replay has completed, perform whatever new debugging steps you want, then select History On a second time to terminate recording. The new tape contains both the original and the added commands.

Note CodeView records only those mouse commands that apply to CodeView. Mouse commands recognized by the application being debugged are not recorded.

Replay Limitations Under OS/2

There are some limitations to dynamic replay when debugging under OS/2:

- The program must not respond to asynchronous events. (Replay under OS/2 Presentation Manager is not currently supported because it violates this restriction.)
- Breakpoints must be specified at specific source lines or for specific symbols (rather than by absolute addresses), or replay may fail.
- Single-thread programs behave normally during replay. However, one of the threads in a multithread program may cause an asynchronous event, violating the first restriction. Multithread programs are, therefore, more likely to fail during replay.
- Multiprocess replay will fail. Each new process invokes a new CodeView session. The existence of multiple sessions makes it impractical to record the sequence of events if you execute commands in a session other than the original.

Advanced CodeView Techniques

Once you are comfortable displaying and changing variables, stepping through the program, and using dynamic replay, you may want to experiment with advanced techniques.

Setting Command-Line Arguments

If your program retrieves command-line arguments, you can specify them with the Set Runtime Arguments command from the Run menu. Enter the arguments in the Command Line field before you begin execution. (Arguments entered after execution begins cause an automatic restart.)

Multiple Source Windows

You can open two Source windows simultaneously. The windows can display two different sections of the same program, or one can show the high-level listing and the other the assembly language listing. In the latter case, the contents of the windows are synchronized. The next assembly language instruction to be executed matches the next line of source code.

You can move freely between these windows, executing a single line of source code or a single assembly instruction at a time.

Using Breakpoints Efficiently

Breakpoints slow execution when debugging. You can increase CodeView's speed by using the /R command-line switch if you have an 80386-based computer. This switch enables the 386's four debug registers, which support breakpoint checking in hardware rather than in software.

Printing Selected Items

To print all or part of the contents of any window:

1. From the File menu, choose Print.
   CodeView displays the Print dialog box.
2. Select the information to print and whether to append the information to an existing file or to overwrite the file.
   By default, print output is to the file CODEVIEW.LST in the current directory. If you want the output to go to a printer, enter the appropriate device name (such as LPT1: or COM2:) in the To File Name field.
3. Choose OK.

Handling Register Variables

A register variable is stored in one of the microprocessor's registers, rather than in RAM. This speeds access to the variable.

A conventional variable can become a register variable when the compiler stores an often-used variable, such as a loop variable, in a register to speed execution.

Register variables can cause problems during debugging. As with local variables, they are only visible within the function where they are defined. In addition, a register variable may not always be displayed with its current value.

Redirecting CodeView Input and Output

The Command window accepts DOS-like commands that redirect input and output. These commands can also be included on the command line that invokes CodeView. Whatever follows the /C option on the command line is treated as a CodeView command to be immediately executed at startup. For example:

```
CV /c infile; t >outfile myprog
```

This command line redirects input to `infile`, which can contain startup commands for CodeView. When CodeView exhausts all commands in the input file, focus automatically shifts to the Command window. Output is sent to `outfile` and echoed to the Command window. The `t` must precede the `>` command for output to be sent to the Command window.

Redirection is a useful way to automate CodeView startup. It also lets you keep a viewable record of command-line input and output, a feature not available with dynamic replay. (No record is kept of mouse operations.) Some applications (particularly interactive ones) may need modification to allow for redirection of input to the application itself.

Customizing CodeView with the TOOLS.INI File

The TOOLS.INI file customizes the behavior and user interface of several Microsoft products. You should place it in a directory pointed to the INIT environment variable. (If you do not use the INIT environment variable, CodeView looks for TOOLS.INI only in its source directory.)

The CodeView section of TOOLS.INI is preceded by the following line:

```
[cv]
```

If you are running the protected-mode version of CodeView, use [cvp] instead. If you run both versions, include both: [cv cvp].

Most of the TOOLS.INI customizations control screen colors, but you can also specify options, such as startup commands or the name of the file receiving CodeView output. Online Help contains full information about all TOOLS.INI switches for CodeView.

Controlling CodeView with Command-Line Options

The following options can be added to the command line that invokes CodeView:

Option	Effect
/2	Two-monitor debugging. One display shows the output of the application; the other shows CodeView.
/25	Display in 25-line mode.
/43	Display in 43-line mode. (EGA or VGA only)
/50	Display in 50-line mode. (VGA only)
/B	Display in black and white.
/C*commands*	All items following this switch are treated as CodeView commands to be executed immediately upon startup. Commands must be separated with a semicolon (;).
/D[*buffersize*]	Use disk overlays, where *buffersize* is the decimal size of the overlay buffer, in kilobytes. The acceptable range is 16K to 128K, with the default size 64K. (DOS only)
/E	Use expanded memory for symbolic information. (DOS only)
/F	Flip screen video pages. When your application does not use graphics, up to eight video screen pages are available. Switching from CodeView to the output screen is accomplished more quickly than swapping (/S) by directly selecting the appropriate video page. Cannot be used with /S. (DOS only)
/I[0 | 1]	/I0 enables nonmaskable-interrupt and 8259-interrupt trapping. This enables Ctrl+C and Ctrl+Break for PCs that CodeView does not recognize as IBM-compatible. /I or /I1 turns off interrupt trapping. (DOS only)
/K	Use this option if you encounter a deadlock situation because the keyboard buffer is full when you restart the program you are debugging, exit CodeView before your program finishes execution, or debug an application that does not accept keystrokes.
/L*dlls*	Load DLLs specified. DLLs must be separated by a semicolon (;). (OS/2 only)
/M	Disable the mouse for CodeView. Does not affect the use of the mouse in the application being debugged.
/N[0 | 1]	/N0 enables nonmaskable-interrupt trapping. This enables Ctrl+C and Ctrl+Break for PCs that CodeView does not recognize as IBM-compatible. /N or /N1 turns off interrupt trapping. (DOS only)
/O	Debug child processes (*offspring*). (OS/2 only)

/R	Use 386 hardware debug registers. (DOS only)
/S	Swap screen in buffers. When your program uses graphics, all eight screen buffers must be used. Switching from CodeView to the output screen is accomplished by saving the previous screen in a buffer. Cannot be used with /F. (DOS only)
/X	Use extended memory for symbolic information. (DOS only)

Chapter 6
Using Online Help

The Microsoft Advisor online Help system is more than a learning tool—it's a reference database designed especially for professional programmers. Help gives you instant reference information about the BASIC language, development environments, compiler options, utilities, and error messages. You can also copy sample code from a Help screen and paste it directly into your BASIC source file. Using the Microsoft Help File Creation utility (HELPMAKE), you can even create your own Help files.

This chapter describes:

- Installing Help.
- Organization of Help.
- Navigating in Help.
- Copying and pasting Help information.
- Creating your own Help files.
- Using QuickHelp.

Installing Help

Use the Setup program to install Help. In the Specify Files to Install menu, turn on the Help Files for Chosen Tools check box. Setup includes all the Help files you need when it installs Microsoft BASIC.

At this point, you can use online Help in the QuickBASIC Extended (QBX) environment. To use Help in the Programmer's WorkBench (PWB), CodeView, and QuickHelp environments, you must specify where each of these applications should look for their Help files by setting the HELPFILES environment variable. HELPFILES can be set in several different places. PWB, CodeView, and QuickHelp search the following locations in the order shown and use the first HELPFILES setting they find:

1. Environment selection from the PWB Options menu. (This only has an effect in PWB.)
2. TOOLS.INI file.
3. AUTOEXEC.BAT file (DOS) or CONFIG.SYS file (OS/2).

If HELPFILES is not set in one of these locations, PWB and QuickHelp will search for help files in the current directory and in the directories listed in the PATH environment variable.

You can set HELPFILES and other environment variables for a session by running the NEW-VARS.BAT (DOS) or NEW-VARS.CMD (OS/2) file from the command line. You may want to incorporate the settings in these files into your AUTOEXEC.BAT (DOS) or CONFIG.SYS (OS/2) files. For information on how to do this, see the section "After Running Setup" in Chapter 2.

Help Topics and Hyperlinks

Online Help is organized as a set of topics. Instead of thumbing through a manual, you can move from one topic to another related topic by using links (or cross-references). Help remembers the last 20 screens you've viewed, so you can retrace your path at any time.

Explicit Cross-References

Explicit cross-references, called "hyperlinks," are tied to a word or phrase at a specific location in Help. You simply select a hyperlink, using the mouse or keyboard, to move to a new screen of related information.

Special characters set off hyperlink buttons. Figure 6.1 shows examples.

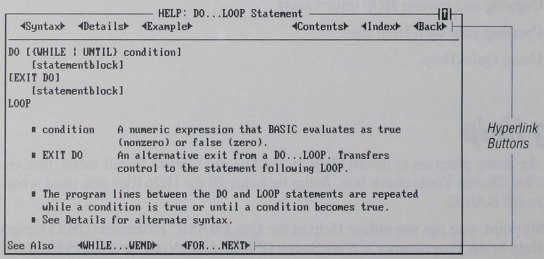

Figure 6.1 Help Screen with Hyperlinks

To move forward between hyperlinks, press the Tab key; Shift+Tab moves you backward. You can also type the first letter of a hyperlink to move to that hyperlink.

Implicit Cross-References

All BASIC keywords (functions, statements, operators, and metacommands) are implicit cross-references. Implicit cross-references have no special formatting. For example, the word "open" provides context-sensitive help on the **OPEN** statement wherever it appears, whether in your programs or in a Help screen. To use an implicit cross-reference, place the cursor on or immediately following the keyword, and then press F1 or click the right mouse button.

Help Categories

There are five categories of help:

- Using Help
- Keyword and symbol Help
- Topic-based Help
- Programming environment Help
- Error message Help

Using Help

You can get information about the online Help system itself at any time by choosing the Using Help command from the Help menu. The Using Help screen provides a quick overview of the Help system and gives instructions for accessing Help with the keyboard or mouse.

Keyword and Symbol Help

Online Help provides context-sensitive help for BASIC keywords and for symbols in your program, such as procedures and variables.

Keyword Help

Keywords include functions, statements, operators, and metacommands. Every keyword in Microsoft BASIC is an implicit cross-reference. To get information on any BASIC keyword (in your program or within the online Help system), place the cursor on the keyword, and press F1 or click the right mouse button. For example, type **do** in the QBX View window, and then press F1. The Syntax screen for the **DO...LOOP** statement appears, as shown in Figure 6.2.

```
HELP: DO...LOOP Statement
 ◄Syntax►  ◄Details►  ◄Example►                ◄Contents►  ◄Index►  ◄Back►

DO [{WHILE | UNTIL} condition]
   [statementblock]
[EXIT DO]
   [statementblock]
LOOP

   ▪ condition    A numeric expression that BASIC evaluates as true
                  (nonzero) or false (zero).
   ▪ EXIT DO      An alternative exit from a DO...LOOP. Transfers
                  control to the statement following LOOP.

   ▪ The program lines between the DO and LOOP statements are repeated
     while a condition is true or until a condition becomes true.
   ▪ See Details for alternate syntax.

See Also    ◄WHILE...WEND►    ◄FOR...NEXT►
```

Figure 6.2 BASIC Syntax Help Screen

If you want to see a programming example using the **DO...LOOP** statement, move to the Help window, then choose the Example hyperlink. If you need additional information, choose the Details hyperlink.

Note You can browse the complete list of BASIC keywords by choosing the Keywords by Task hyperlink from the BASIC Help Table of Contents screen.

Symbol Help

To get information on any symbol in your program from QBX, position the cursor anywhere on the symbol and then press F1 or click the right mouse button. The Help system will identify the symbol (variable, function, etc.) and where it is used in your program. This type of Help is only available in QBX.

To view symbol information in PWB, you must use the Source Browser. For instructions on using the Source Browser, see Chapter 4, "Using PWB."

Topic-Based Help

You can use the Help system to get an overview of available topics when you have a general idea of the information you need. For example, suppose you want to learn about QBX's customizable keystrokes, but you don't know where to look. Choose the Contents hyperlink in any Help window, or choose Contents from the Help menu. The BASIC Help Table of Contents screen appears, as shown in Figure 6.3.

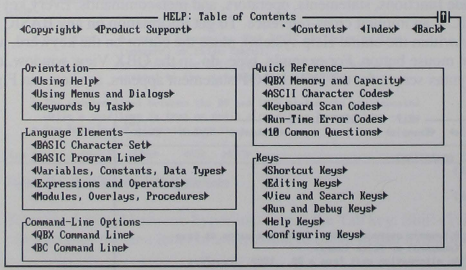

Figure 6.3 BASIC Help Table of Contents Screen

The BASIC Help Table of Contents screen is broken into five subgroups: Orientation, Language Elements, Command-Line Options, Quick Reference, and Keys.

Programming Environment Help

You can get context-sensitive Help on any menu command or dialog box. For example, to learn about the Create File command in the QBX File menu:

1. Open the File menu.
2. Press the Down direction key to highlight Create File (do not press Enter), and then press F1. The Help screen for the Create File command appears, as shown in Figure 6.4.

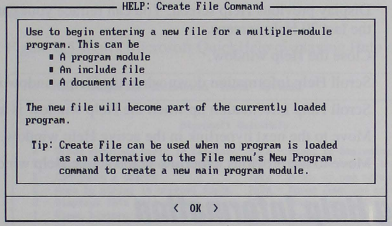

Figure 6.4 Help on the Programming Environment

You can also get help about a dialog box. Press F1 or click the Help button to display information about the dialog box.

Error Message Help

Whenever you encounter an error message, you can press F1 to get help on the cause of the error. Most error message Help screens contain suggestions on how to avoid the error.

Navigating in the Help System

The following table summarizes Help keystrokes.

Press this key	To do this
F1	Get Help on the selected item.
Shift+F1	In QBX, get help on using the Help system. In PWB, CodeView, and QuickHelp, Shift+F1 displays the Help Table of Contents.
Alt+F1	Display previous Help screen. You can retrace your path through the last 20 Help screens.
Esc	Close the Help window.
PgDn	Scroll Help information down when the Help window is active.
PgUp	Scroll Help information up when the Help window is active.
Tab	Move to the next hyperlink in the active Help window.
Shift+Tab	Move to the previous hyperlink in the active Help window.

Copying and Pasting Help Information

You can copy any text that appears in the Help window to another window. To test a sample program from the Help window, just copy it to the program window and then choose Start (QBX) or Execute (PWB) from the Run menu.

To copy and paste, follow these steps:

1. Select the text you want to copy.
2. Press Ctrl+Ins.
3. In the program window, place the cursor where you want to insert the text, and then press Shift+Ins.

Creating Custom Help Files

The HELPMAKE utility allows you to create or modify Help files for use with Microsoft products. For example, you could write Help text describing a new function you have written for a Quick library. You can then display Help information about your function just as you would display Microsoft-supplied Help information.

HELPMAKE translates Help text files into a Help database accessible from the QBX programming environment or other Microsoft language products. You can either create Help files from scratch, or you can use HELPMAKE to decompress the Microsoft-supplied Help database and then modify it.

For complete information on building your own Help system, see Chapter 22, "Customizing Online Help," in the *Programmer's Guide*, or select HELPMAKE from the main Help Table of Contents in PWB.

Using QuickHelp

Microsoft QuickHelp lets you read Microsoft Advisor Help files without running a programming environment. This is useful if you are running Microsoft Windows or OS/2 and you want to keep Help information open in a window at all times. QuickHelp is also useful for reading Help databases that you have created for your own applications.

To start QuickHelp, type:

```
QH
```

Figure 6.5 shows Microsoft QuickHelp displaying Help on itself.

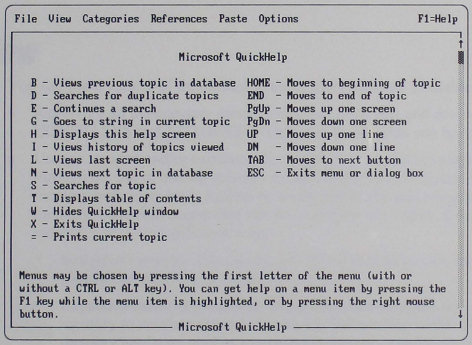

Figure 6.5 QuickHelp

QuickHelp searches for Help files in the paths and the order described in the section "Installing Help" earlier in this chapter. In addition, QuickHelp also uses the QH environment variable to define additional directories to search. For example:

```
SET HELPFILES=C:\BC7\HELP
SET QH=C:\LANMAN\HELP
```

If these lines are entered from the command line, QuickHelp will search in C:\BC7\HELP and C:\LANMAN\HELP for Help files. PWB and CodeView, however, will not search in C:\LANMAN\HELP unless it is included in the HELPFILES environment variable.

Chapter 7
Memory Management in DOS

This chapter explains how to increase the amount of memory available under DOS to help you create the largest possible running applications using each of the Microsoft BASIC environments. It shows you how to take advantage of extended and expanded memory by using one or more of the three device drivers included in the package: HIMEM.SYS, SMARTDRV.SYS, or RAMDRIVE.SYS.

Overview

You can use a variety of techniques to make more memory available for program use, such as:

- Removing all unnecessary memory-resident programs and device drivers. (Check your CONFIG.SYS and AUTOEXEC.BAT files to see what you have installed.)
- Using extended and/or expanded memory. Extended and expanded memory are available through the use of add-on memory boards. Some system boards have extended or expanded memory built in as well. Add-on memory boards include extended memory, expanded memory, and some boards that can be configured for either. Expanded and extended memory are used in different ways:
  - Extended memory. You can use the HIMEM.SYS device driver provided with this package or another extended-memory driver.
  - Expanded-memory emulators. Many extended-memory drivers can use some or all memory to emulate expanded memory. Information in this chapter about expanded-memory hardware applies to expanded-memory emulators as well.
  - Expanded memory. Expanded memory provides up to 1.2 megabytes of buffer space to the ISAM portion of a BASIC program.

  You can use extended and expanded memory at the same time. However, only one expanded-memory driver and one extended-memory manager can run on the system. Use an expanded-memory emulator only if you have no expanded-memory hardware.

Using Extended Memory (XMS)

Extended memory requires an 80286 or later processor. The extended-memory device drivers compatible with BASIC observe the Extended Memory Specification (XMS), version 2.0 and higher. You can use the HIMEM.SYS device driver provided in this package, or you can use your own extended-memory driver if it supports XMS version 2.0 or higher.

The HIMEM.SYS Device Driver

The HIMEM.SYS device driver provides a 64K area immediately above one megabyte called the High Memory Area (HMA). Only one piece of software can reside in HMA.

You must have DOS version 3.0 or later to run HIMEM.SYS. To install HIMEM.SYS, place the following statement in your CONFIG.SYS file:

device=[[*drive:*][*path*]HIMEM.SYS [/HMAMIN=*tsrmin*][/NUMHANDLES=*handles*] [/SHADOW=[ON | OFF]] [/MACHINE=*machine*]

In the preceding syntax, *tsrmin* is the minimum size, in kilobytes, that a memory-resident program must have to be placed in HMA. The default value of 0 lets any memory-resident program request use of this area for extended memory.

The *handles* field sets the maximum number of Extended Memory Block handles that HIMEM.SYS supports. Each piece of software that you load into extended memory above HMA requires an additional handle. The *handles* field defaults to 32 and must be between 1 and 128. Each additional handle requires 6 bytes of memory.

The /SHADOW option enables or disables shadow RAM. Some computers make ROM faster by "shadowing" it in RAM — that is, copying the ROM code into RAM memory at startup. This uses some extended memory. On systems with 384K of extended memory, HIMEM.SYS automatically disables shadowing. You can use /SHADOW=ON to explicitly enable shadowing on systems with less than 384K of extended memory.

The /MACHINE option tells HIMEM.SYS what type of computer you are using. In most cases, HIMEM.SYS automatically detects the type of machine you are using. Currently, the only system which requires this switch is the Acer 1100. If you are using this system, you should use the option /MACHINE:ACER1100.

Other Extended-Memory Device Drivers

In general, extended-memory device drivers make memory available by using three kinds of addresses:

- A 64K area starting at address FFFF:0010, just above conventional memory.
- Addresses above this 64K area. This region can be accessed by temporarily switching into protected mode, then copying the contents to a lower address.
- Unused addresses above 640K but below one megabyte. These are areas reserved by the system but currently unused.

HIMEM.SYS supports the first two kinds of addressing.

Some extended-memory device drivers can move memory-resident programs into the third area—unused addresses below one megabyte. Doing so makes more conventional memory available.

For example, if you have an 80386 processor and the 386MAX.SYS device driver (from Qualitas, Inc.), you may be able to move the ISAM TSR into unused address space. This requires a sufficiently large contiguous area of memory and is dependent on your system configuration. Install 386MAX.SYS in your CONFIG.SYS file, and then execute the following DOS-level commands:

```
386MAX loadhigh
PROISAMD
386MAX loadlow
```

The first command directs the 386MAX driver to load software into high memory (above 640K). PROISAMD is the full version of the ISAM memory-resident program (database creation and access). The third command restores normal loading.

Using Expanded Memory (EMS)

Expanded memory is compatible with all 8086-family processors. It is implemented by a paging mechanism to swap up to 64K into an 8086-addressable area. The expanded-memory device drivers compatible with BASIC observe the Lotus-Intel-Microsoft (LIM) Expanded Memory Specification (EMS). To ensure compatibility with QBX and ISAM, use device drivers that support the LIM 4.0 specification.

To use expanded memory, you must install an expanded-memory device driver in your CONFIG.SYS file. Hardware manufacturers supply a device driver with their expanded memory board. Two of the device drivers included in this package, RAMDRIVE.SYS and SMARTDRV.SYS, can be used with expanded memory.

Note All of the features described in this section also apply to expanded-memory emulators.

Using RAMDRIVE

The RAMDRIVE.SYS device driver lets you use a portion of your computer's memory as an additional hard disk. This disk is referred to as a RAM disk or virtual disk and is much faster than a physical hard disk.

You can place a RAM disk in conventional, extended, or expanded memory. On most machines, RAMDRIVE.SYS runs fastest with expanded memory. RAMDRIVE.SYS always uses some conventional memory. The exact amount depends on your hardware configuration and varies between systems.

The system reads and writes to the RAM disk almost as fast as to main memory. If you set the TMP environment variable to a directory on a RAM disk, a number of programming tools (notably LINK and ISAM) will use the RAM disk to hold temporary files, thus speeding up operations. You can also copy libraries and frequently used executable files to RAMdrive at the beginning of a session to increase speed. Placing source files on the RAM disk is risky since all information is deleted from the RAM disk when you reboot.

To install RAMDRIVE.SYS, include the following command line in your CONFIG.SYS file:

device=[*drive:*][*path*]RAMDRIVE.SYS [*disksize*][*sectorsize*][*entries*][{/A|/E|/U}]

The following list explains the meaning of each option:

Option	Description
disksize	Disk size in kilobytes. The default size is 64K, the minimum is 16K, and the maximum is 4096K.
sectorsize	Sector size in bytes. The default size is 512 bytes. The other legal values are 128, 256, and 1024. If in doubt, use the default.
entries	Limits the number of root directory entries (files and subdirectories). The default size is 64, the minimum is 2, and the maximum is 1024.
/A	Uses expanded memory. (This memory must conform to the LIM 4.0 specification.) If you use /A, you cannot use /E or /U.
/E	Uses extended memory, making use of addresses above the one-megabyte address space. If you use /E, you cannot use /A or /U.
/U	Applicable only to AT&T 6300 PLUS motherboard. Specifies that some or all of the 384K of extra memory on an AT&T 6300 PLUS motherboard is to be used as an extra RAM drive. 1K of the 384K is reserved as overhead for RAMDRIVE.SYS. If you use /U, you cannot use /A or /E.

Note

Any use of RAMDRIVE.SYS or SMARTDRV.SYS uses some conventional memory.

The versions of RAMDRIVE.SYS and SMARDRV.SYS provided with BASIC are compatible with Windows version 3.0. They are not compatible with Windows versions 2.11 and earlier.

Using SMARTDrive

The SMARTDRV.SYS device driver lets you use a portion of your computer's memory as a disk-cache area. This mechanism serves as a temporary holding area for recently accessed data from the hard disk, which significantly reduces data-access time if the desired information is in cache.

You can place the SMARTDrive disk cache in either extended or expanded memory.

To install SMARTDRV.SYS, include the following command line in your CONFIG.SYS file:

device=[[*drive:*]][[*path*]]SMARTDRV.SYS [[*normal cachesize*]][[*minimum cachesize*]][[/A]]

The following list explains the meaning of each option:

Option	Description
normalcachesize	Size of the disk cache in kilobytes. The default size is 256K for extended memory and all of available memory for expanded memory. Several megabytes is a typical size for a disk cache; you'll generally see little difference in performance unless the cache is well over 500K in size.
minimumcachesize	The smallest disk cache in kilobytes. Windows will reduces disk cache to this number when running in real mode.to make more memory available for its own use.
/A	Uses expanded memory. If this option is not present, then SMARTDrive attempts to use extended memory.

Generally, the memory assigned to SMARTDRV.SYS cannot be used by other software. However, CodeView and Windows can share memory with SMARTDrive. Any remaining memory is used for disk-cache operations.

ISAM Use of Expanded Memory

By default, ISAM uses about 1.2 megabytes of expanded memory for buffer space (or as much as available, whichever is less). This helps to free conventional memory and improves performance by providing far more buffers than could exist in conventional memory.

Depending on how much expanded memory you have, ISAM's use of 1.2 megabytes may affect the performance of other programs that need this memory. To limit the amount of expanded memory that ISAM uses, use the /Ie command-line option with the BASIC Compiler or when starting the ISAM memory-resident program:

/Ie:*emsreserve*

In the syntax above, *emsreserve* is the number of kilobytes to reserve for other software. The setting /Ie:-1 is a special value that disables ISAM's use of expanded memory altogether.

For example, the following command line starts the ISAM memory-resident program and reserves 800K of expanded memory for other software, such as QBX or CodeView:

```
PROISAMD /Ie:800
```

Use /Ie in conjunction with the QBX /Es option if your program code or a loaded Quick library manages expanded memory. (See "Competing Use of Expanded Memory" later in this chapter.)

Several other options affect how ISAM handles buffers. For complete documentation on how to configure and install ISAM, as well as how the number of buffers affects programs, see Chapter 10, "Database Programming with ISAM," in the *Programmer's Guide*.

Considerations for QBX

The QBX environment and ISAM together require 450K to run (QBX itself requires over 300K). If you have 640K of conventional memory, QBX will support a moderate-sized program without trouble, but large programs may require the use of extended or expanded memory, especially if you use ISAM.

Using the /NOFRILLS Option

You can use the /NOFRILLS option (abbreviated as /NOF) to make additional memory available for program use. However, using /NOFRILLS reduces the functionality of the QBX environment.

When you start QBX and use the /NOF command-line option, QBX uses about 19K less memory but does not support any of the following menus:

- Utility menu
- Options menu
- Help menu

The /NOF option does not change the structure of the menu bar or status line, but the three menus are unavailable and cannot be opened.

Extended Memory

QBX uses extended memory by moving approximately 60K of its own executable code out of conventional memory. All the code removed represents space freed for use by your programs.

When using the HIMEM.SYS device driver, you should set *tsrmin* to 63 (64,512 bytes) to ensure that QBX has access to the high-memory area. The *handles* field has no effect on QBX, since these handles give access to addresses higher than the 64K area which QBX can use.

For example, the following entry in your CONFIG.SYS file installs HIMEM.SYS and ensures that extended memory is available for QBX:

```
device=C:\DEVICE\HIMEM.SYS /HMAMIN=63
```

Expanded Memory

When you start the QBX environment, expanded memory may have already been allocated to memory-resident programs, most notably ISAM. QBX uses as much of the the remaining expanded memory as it needs.

You can limit the amount of expanded memory used by QBX by using the /E: command-line option:

/E:*emslimit*

In the preceding syntax, *emslimit* determines the maximum amount of expanded memory that QBX will use, in kilobytes. A value of 0 disables QBX usage of expanded memory and can improve the execution speed of some operations.

Assuming some expanded memory is available, QBX takes advantage of the memory in two ways:

- QBX automatically moves each unit of program code smaller than 16K into expanded memory. Each procedure is a unit of code. The module-level code also constitutes a unit of code.

  To see how large each procedure is, use the View SUBs command (press F2). This command lists each unit of code together with size in kilobytes, rounded to the nearest kilobyte. You can determine which units are larger than 16K and should be divided into smaller units (see Figure 7.1). To guarantee best results, View SUBs should show sizes ranging from 0 to 15, inclusive.

- If you give the /Ea command-line option, QBX moves arrays into expanded memory if they do not contain variable-length strings and are smaller than 16K. This option has a drawback: arrays residing in expanded memory cannot be passed to modules written in other languages (although you can pass array elements). Therefore, you may not want to use /Ea with mixed-language programs.

  This option affects all types of arrays except arrays of variable-length strings, which are not placed in expanded memory.

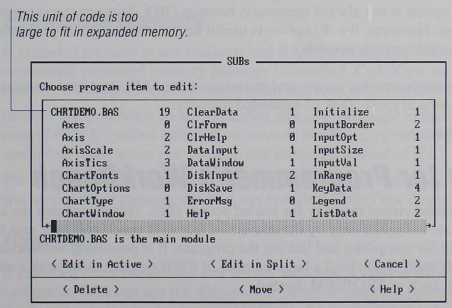

Figure 7.1 QBX View SUBS Dialog Box

The following QBX command line allows 500K of expanded memory to be used and moves arrays out of conventional memory:

```
QBX /E:500 /Ea
```

Note Once inside the QBX environment, you can determine how much expanded memory is actually available to QBX by executing the statement FRE(-3).

Competing Use of Expanded Memory

A Quick library can make use of expanded memory directly by using assembly code that calls the expanded-memory interrupt. However, Quick libraries that do this must compete with QBX for use of the memory. QBX will use as many 16K pages of expanded memory as required, according to the rules described in the previous section (assuming that the memory is available).

In order to use expanded memory for Quick libraries, you need to keep two things in mind:

- You must use the /Es command-line option when starting QBX. This option saves and restores the EMS state before and after every ISAM statement or call to a Quick library. This save-and-restore operation involves significant overhead, so don't use it unless you have a Quick library that manages expanded memory.
- You may want to limit QBX use of expanded memory by using the /E: command-line option. This option is not always necessary, because QBX does not reserve more memory than it can use. However, the /E: option is useful for guaranteeing that your program always has the expanded memory it needs.

The following example enables saving and restoring of the EMS state, and restricts the QBX environment to using at most 500K of expanded memory:

```
QBX /E:500 /Es
```

Considerations for Programmer's WorkBench

PWB uses disk-based virtual memory for loaded source files. This means that the size of the program you create is limited only by the amount of free disk space you have and the amount of memory available for compiling and linking the program. It is possible, however, to run out of memory for PWB itself if you load a large number of PWB extensions or have a memory-resident program, such as PROISAM, loaded.

You can make more memory available in PWB by not automatically loading the extension files. The /DA options disables the auto-load function. You can then explicitly load only the extensions you need by customizing your TOOLS.INI file. For more information, see the section "Loading PWB Extensions Explicitly" in Chapter 4.

Considerations for CodeView

When you debug a program in CodeView, three components compete for memory:

- The program being debugged
- The program's symbolic information
- CodeView itself

To make the most memory available to the executable being debugged, CodeView automatically uses expanded memory, extended memory, or disk overlays. In addition, CodeView provides a utility for compressing program symbol information (CVPACK). For more information on CVPACK, see Chapter 5, "Using CodeView."

Extended Memory

If HIMEM.SYS or another extended-memory device driver is installed, CodeView automatically places all of the program symbol information and all but 16K of CodeView itself in extended memory, leaving the remaining conventional memory available for program execution.

The /X option is not required to use extended memory with CodeView, but if you specify /X and an extended memory driver is not installed, CodeView displays an error.

Expanded Memory

If extended memory is not available and RAMDRIVE.SYS. SMARTDRV.SYS, or a compatible expanded memory manager is installed, CodeView uses available expanded memory for program symbol information and for CodeView overlays.

In order for CodeView to use expanded memory, no single module's symbol information can exceed 48K. If symbol information exceeds this limit, try reducing filename information by not specifying paths when you compile and using /Zd instead of /Zi.

The /E option is not required to use expanded memory with CodeView, but if you specify /E and an expanded memory driver is not installed, CodeView displays an error.

Overlays

If extended or expanded memory are not available, CodeView uses 64K disk overlays for its executable. You can use the /D*buffersize* option to improve performance by reducing the size of the overlay, making more memory available, or increasing the size of the overlay.

CodeView will not use overlays smaller than 16K nor larger than 128K. Therefore, the minimum effective value for *buffersize* is 16 and the maximum is 128.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Microsoft®
Making it all make sense™

0390 Part No. 12591